VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Metody statické analýzy malware

Methods of the Static Malware Analysis

2020 Bc. Jan Hložek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jan Hložek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 1801T064 Informační a komunikační bezpečnost

Téma: Metody statické analýzy malware
Methods of the Static Malware Analysis

Jazyk vypracování: čeština

Zásady pro vypracování:

Práce je zaměřena na pochopení principů vybraných metod statické analýzy malware. Cílem a účelem práce je vytvořit sadu ukázkových případových studií vysvětlující statickou analýzu viru didaktickým způsobem. Student také implementuje program na statickou analýzu malware zahrnující aktuální metody.

Předpokládaná struktura práce je:
1. Seznámení se s problematikou.
2. Volba vhodného programovacího prostředí.
3. Volba vhodných postupů a algoritmů z oblasti statické analýzy.
4. Programová realizace těchto algoritmů.
5. Analýza vybraných virů jak v práci vytvořených, tak již existujících vzorků.
6. Tvorba uživatelského manuálu.

Seznam doporučené odborné literatury:

[1] Merhaut F., Zelinka I., Úvod do počítačové bezpečnosti, Fakulta aplikované informatiky, UTB ve Zlíně, Zlín, 2009
[2] Peter Szor, Počítačové viry - analýza útoku a obrana, Zoner Press
[3] Zelinka I., Oplatková Z., Šeda M., Ošmera P., Včelař F., Evolutionary techniques – principles and applications, BEN, Prague, 2008, 598 p.
[4] Kumar, Vipin, Jaideep Srivastava, and Aleksandar Lazarevic, eds. Managing cyber threats: issues, approaches, and challenges. Vol. 5. Springer Science & Business Media, 2006.
[5] Singer, Peter W., and Allan Friedman. Cybersecurity: What Everyone Needs to Know. Oxford University Press, 2014.
[6] Moshchuk, Alexander, Tanya Bragin, Steven D. Gribble, and Henry M. Levy. "A Crawler-based Study of Spyware in the Web." In NDSS, vol. 1, p. 2. 2006. Harvard
[7] Balthrop, Justin, Stephanie Forrest, Mark EJ Newman, and Matthew M. Williamson. "Technological networks and the spread of computer viruses." Science 304, no. 5670 (2004): 527-529.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **prof. Ing. Ivan Zelinka, Ph.D.**

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2020

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 30. dubna 2020 ...

Rád bych poděkoval prof. Ing. Ivanu Zelinkovi, Ph.D. za věcné poznámky a odborné vedení mé diplomové práce.

Abstrakt

Autor se v této diplomové práci zabývá především statickou analýzou. Na základě předložené rešerše obsahující jak informace o analýze malwaru a současných trendech v této oblasti, tak i možnostech obrany malwaru před takovou analýzou, nabízí své řešení pomocí statické analýzy. Své řešení poté dále testuje se sadou vzorků legitimního softwaru a malwaru. Výstupem pak je řada parametrů jako jsou například kódem použité nástroje, knihovny, řetězce, entropie kódu nebo výskyt použitých pravidel Yara. Na základě těchto výstupů pak autor dokazuje, že jím předložené řešení je funkční.

Klíčová slova: statická analýza, analýza malwaru, kybernetická obrana, formát spustitelných souborů, ochrana spustitelných souborů

Abstract

In this diploma thesis, the author focuses mainly on static analysis. Based on the submitted research of literature, which contains information about the analysis of malware and current trends in this area, as well as the possibilities of defending malware against such analysis, the author offers his own solution using static analysis. His solution is then further tested with set of samples of legitimate software and malware. The output is several parameters such as the code of the tool used, the library, strings, the entropy of the code or the occurrence of the Yara rules used. Based on these outputs, the author proves that the solution presented by him is functional.

Keywords: static analysis, malware analysis, cyber defence, executable file formats, executable file protection

Obsah

Seznam použitých zkratek a symbolů **10**

Seznam obrázků **11**

Seznam tabulek **12**

Seznam výpisů zdrojového kódu **13**

1 Úvod **14**

2 Motivace **15**

3 Analýza malwaru **16**
3.1 Statická analýza 16
3.2 Dynamická analýza 17

4 Formát spustitelných souborů **18**
4.1 Spustitelné soubory v OS 18
4.2 PE hlavička 19

5 Základní nástroje statické analýzy **24**
5.1 Debugger 24
5.2 Disassembler 24
5.3 HEX editor 25
5.4 Balík sysinternals 25
5.5 Dependency Walker 26

6 Obranné mechanismy proti analýze softwaru **27**
6.1 Anti-debug 27
6.2 Obfuskace 28
6.3 Komprese 33
6.4 Packery 33
6.5 Unpacking 34
6.6 Anti-VM 35

7 Současný stav poznání **39**
7.1 Statická a dynamická analýza malware pomocí strojového učení 39
7.2 Klasifikace malwaru na základě charakteristik získaných pomocí statické analýzy 39
7.3 Statická analýza malwaru v systému android: techniky, limity a přetrvávající výzvy 39
7.4 Limity statické analýzy pro detekci malwaru 40

7.5 Analýza výkonu strojového učení a algoritmu pro rozpoznávání schémat pro klasifikaci malwaru . . . 40
7.6 Technologické sítě a šíření počítačových virů . . . 41
7.7 Analýza a klasifikace malwaru: průzkum . . . 41

8 Experimentální část **42**
8.1 Použité nástroje a knihovny . . . 42
8.2 Popis aplikace . . . 43
8.3 Výstup aplikace . . . 47
8.4 Testování aplikace . . . 53

9 Závěr **63**

Literatura **65**

Přílohy **71**

A Příloha v IS EDISON **72**

B Jednotlivé hashe **73**
B.1 Malware . . . 73
B.2 Software . . . 74

Seznam použitých zkratek a symbolů

IEEE	–	Institute of Electrical and Electronics Engineers
CnC	–	Command-and-Control server
VBox	–	VirtualBox
OS	–	Operační systém
VM	–	Virtual Machine
HW	–	Hardware
SW	–	Software
MIME	–	Multipurpose Internet Mail Extensions
XOR	–	Exkluzivní disjunkce
PE	–	Portable Executable
ELF	–	Executable and Linkable Format
PEB	–	Process Environment Block
TLS	–	Thread Local Storage
ABI	–	Application binary interface
PE	–	Portable Executable
NE	–	New Executable
ELF	–	Executable and Linkable Format
COFF	–	Common Object File Format
ČSÚ	–	Český statistický úřad
EP	–	Entry Point
API	–	Application Programming Interface
XML	–	Extensible Markup Language
PHP	–	Hypertext Preprocessor
GUI	–	Graphic User Interface

Seznam obrázků

1 Počet vzorků nového malwaru mezi roky 2007 - 2018. Zdroj dat [11] 15
2 Proces spuštění souboru 18
3 PE formát 20
4 Program PEiD 22
5 Program Exeinfo PE 22
6 Nástroj PE Studio 23
7 Debugger OllyDbg 24
8 HEX editor FileInsight 25
9 Dependency Walker 26
10 De/šifrování pomocí XOR 30
11 Jednotlivé vrstvy aplikace 44
12 Ukázka výstupu informací o spustitelném souboru formátu PE 46
13 Ukázka výstupu konzolové aplikace 48
14 Práce s konzolovou aplikací - spuštění 49
15 Práce s konzolovou aplikací - příkaz help 49
16 Práce s konzolovou aplikací - test pomocí Yara 50
17 Práce s konzolovou aplikací - odeslání na VirusTotal 50
18 Ukázka výstupu webové aplikace 51
19 Práce s webovou aplikací krok 1. 52
20 Práce s webovou aplikací krok 2. 52
21 Práce s webovou aplikací krok 3. 53
22 Entropie vzorků 55

Seznam tabulek

1 Formáty spustitelných souborů (Zdroj dat [21]) 19
2 Výběr některých packerů 34
3 Názvy získaných hypervizorů pomocí funkce CPUID 36
4 Některé MAC adresy, které se používají ve virtuálním prostředí - převzato z [1] . 37
5 Ukázka entropie packerů UPX a ASPack 46
6 Podíl spustitelných souborů v datasetu 54
7 Parametry spustitelných souborů v kolekci dat 54
8 Počty sekcí v jednotlivých skupinách 55
9 Entropie jednotlivých skupin 55
10 Výskyt konkrétního typu řetězce 56
11 Porovnání výskytu použitých knihoven 56
12 Porovnání výskytu použitých metod pro malware a software 57
13 Porovnání detekovaných YARA pravidel 58
14 Detekce vytvořeného malwaru 60
15 Detekce malwaru z databáze VirusShare.com 61
16 Detekce legitimních aplikaci 62

Seznam výpisů zdrojového kódu

1 Generování statických dat . . . 29
2 Proměnná bez obfuskace . . . 29
3 Obfuskovaná proměnná . . . 29
4 Původní kód funkce bez obfuskace . . . 31
5 Obfuskovaný kód funkce . . . 31
6 Původní kód bez obfuskace vloženého kódu . . . 32
7 Kód funkce po obfuskaci . . . 32
8 Kód před nahrazením . . . 33
9 Kód po nahrazení . . . 33
10 Instrukce CPUID - Převzato z [1] . . . 36
11 Anti VMware - Převzato z [2] . . . 37
12 Powershell kód pro získání instalovaných ovladačů VirtualBoxu . . . 37
13 Seznam hostitelských ovladačů VirtualBoxu . . . 38
14 Seznam ovladačů VBox hosta . . . 38
15 Powershell kód pro získání běžících procesů a služeb . . . 38
16 Výstup konzole hostitele . . . 38
17 Výstup konzole hosta . . . 38
18 Ukázka pravidla YARA . . . 43

1 Úvod

V dnešní době je standardem používat počítač, telefon, tablet a podobná zařízení. A to jak doma, tak i v práci. Podle českého statistického úřadu používalo v roce 2019 více než 80 % Čechů nad 16 let internet (chytrý telefon 70 % Čechů) [3].

S narůstajícím počtem uživatelů internetu roste i množství potenciálních zranitelných cílů. Zároveň pro laiky může být těžké si tuto zranitelnost uvědomovat a to i přes to, že z médií jsou konfrontováni s útoky na společnosti jako je například OKD [4] nebo nemocnice [5]. Podle ČSÚ pak byly v roce 2018 napadeny až dvě pětiny velkých firem [6].

Navíc v aktuální situaci koronavirové pandemie, kdy lidé, kteří mohou, pracují převážně z domova, může narůstat bezpečnostní riziko jejich zaměstnavatelů, protože je komplikovanější zajistit zabezpečení na stejné úrovni jako v interní síti. Také je možné očekávat, že i v budoucnu bude více využívána práce z domova a to jak v zaměstnání, tak i při studiu a to i na nižších stupních vzdělání, kde to donedávna nebylo pravidlem.

Zmíněna rizika mohou být například:

- únik citlivých informací způsobený phishingovým útokem, což je útok často šířený pomocí důvěryhodně vypadající emailové nebo instantní komunikace s cílem získat citlivá data jako přístupové údaje, nebo rootkitem, jenž je škodlivý kód (neboli malware) sloužící útočníkovi, ke vzdálenému přístupu, sběru informací případně modifikaci systémové konfigurace;
- ztráta dat nebo odepření přístupu, které mohou být způsobeny ransomwarem, což je druh malwaru šifrujícího či mazajícího data;
- zneužití výpočetního výkonu uživatele botem (opět druh malwaru), a to za účelem těžby kryptoměn nebo zapojení do počítačové sítě zvané botnet, kde každý účastník vykonává požadované operace na základě instrukcí získaných od záškodníka;
- krádež identity nebo stalking pomocí škodlivého kódu nazývaného spyware, který útočník využívá k nepozorovanému sledování a sběru informací o oběti [7].

Je velmi důležité tyto hrozby včas detekovat a předejít tak ekonomickým a dalším možným dopadům.

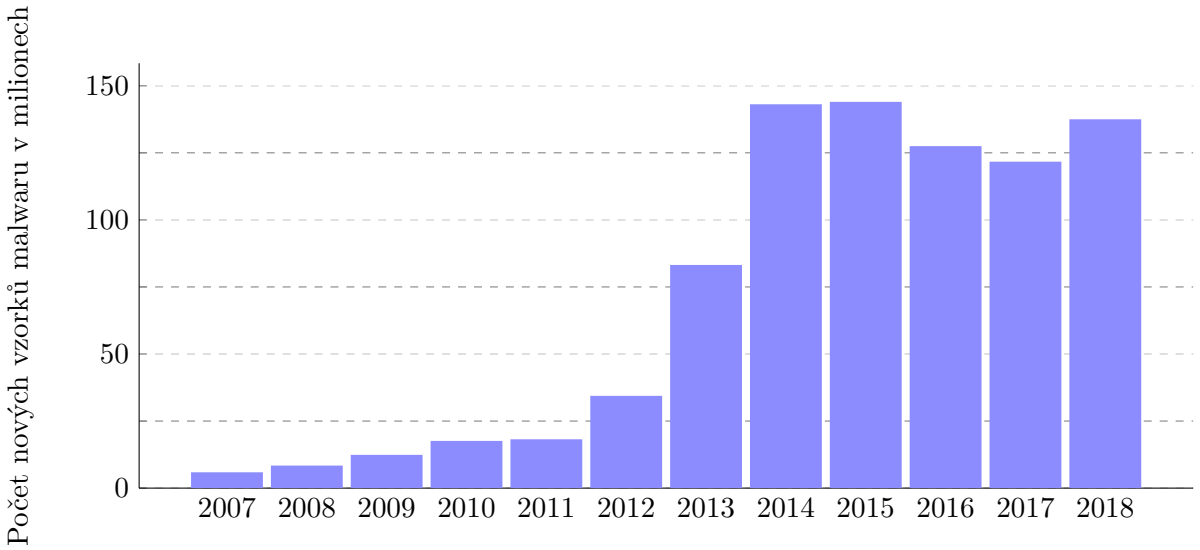
Phishingovým útokům lze do jisté míry předcházet správně nakonfigurovaným obranným softwarem, jenž se snaží komunikaci klasifikovat a odhalit tak tento útok. I přesto se však stále útočníkům daří takové útoky realizovat a získat tak od obětí citlivé informace nebo peníze, jak je možné ukázat na příkladu nemocnice v Sušici [8]. Proto je důležité dostatečně proškolit jedince, kteří mohou být potencionálně zranitelní [9].

O detekci škodlivého kódu se snaží především antivirové společnosti, které vyvíjí produkty pro identifikaci přítomnosti malwaru na základě skenování. Škodlivý kód neboli malware je druh počítačového programu, jehož záměrem je způsobit újmu. Příkladem mohou být jak výše zmíněné ransomwere, spyware nebo rootkit a dále také virus, trojský kůň nebo červ [10].

2 Motivace

Vzhledem k výše zmíněné závažnosti rizik je potřeba reagovat a odhalit nový malware včas, neboť malware už při prvním kontaktu s obětí může způsobit škodu.

Jak je vidět v grafu (viz obrázek č. 1) na následujícím obrázku, množství nového malwaru od roku 2011 rostlo zhruba dvojnásobně, nejvíce nového malwaru pak bylo v roce 2015, kdy se růst počtu nových vzorků prozatím zastavil, nicméně i přesto je jich ročně stále více než 100 milionů.



Obrázek 1: Počet vzorků nového malwaru mezi roky 2007 - 2018. Zdroj dat [11]

Proto jsou vyvíjeny metody automatické analýzy (např. pomocí klasifikace výstupu analýzy kódu a jeho chování viz kapitola 7), jež umožňují získat potřebná data o testovaných vzorcích včas a zmírnit tak případné dopady dosud neodhalené hrozby. Zároveň je také důležité, aby daný poskytovatel anti-malwarových služeb měl zajištěn dostatečný přísun vzorků, které tvoří reprezentativní zástupce. Takto získané vzorky mohou tvořit databázi malwaru.

Aby byl zohledněn vznik množství nových vzorků malwaru a také patřičná rychlost analýzy, je navrhováno množství nových přístupů a postupů při odhalování.

U těchto postupů pak obecně rozlišujeme statickou a dynamickou analýzu, které jsou dále podrobně popsány v teoretické části.

Tato práce se bude dále zabývat především statickou analýzou, možnostmi jejího využití a to i v kombinaci s dalšími metodami, a jejími limity.

3 Analýza malwaru

Analýza malwaru je proces při kterém se zkoumá chování vzorku. Předmětem takovéto analýzy je zjistit, jak malware pracuje a jak jej případně eliminovat [12]. Výstup by měl obsahovat jednotlivé charakteristiky a funkcionality malwaru. Analýza podezřelých binárních souborů se provádí v bezpečném prostředí [13]. Jak již bylo zmíněno, malware může nabývat mnoha podob jako například virus, červ, spyware nebo trojský kůň.

Analýza malwaru je prováděna v následujících případech [12]:

- bezpečnostní incident v síti;
- za účelem zkoumání vzorku;
- indikace kompromitování systému.

Přístupy k analýze jsou dva a to pasivní a aktivní neboli statická a dynamická analýza. Popsány jsou v následujícím textu.

3.1 Statická analýza

Statická analýza je metoda, při níž není zkoumaný vzorek počítačového programu nebo jeho část spuštěn. Tento postup se využívá při analýze malwaru, testování aplikací, hledání zranitelnosti atd. Může se provádět buď ručně nebo pomocí specializovaného programu.

Při statické analýze vzorku je nahlíženo na různé části programu jako jsou metadata, struktura souboru popřípadě zdrojový kód.

Průběh statické analýzy se může lišit případ od případu a nemusí být nutně provedeny všechny kroky [13]. Obvykle však analýza malwaru začíná určením cílové platformy a to na základě informací získaných o spustitelném souboru a z něj. Například pokud podezřelý binární soubor v sobě obsahuje PE hlavičku, s velkou pravděpodobností se jedná o soubor určený pro operační systém Windows.

Malware určený pro Windows často končí příponami .exe, .dll, sys. apod. Spoléhat se pouze na příponu však nelze. Útočník může použít nejrůznější způsoby jak příponu zakrýt tak, aby umožnil malwaru spuštění uživatelem. Proto by v kódu měla být použita signatura souboru, kterou známe také jako magická čísla [14]. Podpis souboru je totiž unikátní sekvence bytu v hlavičce souboru. Různé soubory mají různé signatury a lze je podle nich identifikovat [15].

Dalším krokem je vytvoření unikátního otisku tzv. fingerprintu pomocí kontrolního součtu pro identifikaci daného vzorku a jeho porovnání s databází. Otisk souboru může být vytvořen pomocí libovolné hashovací funkce, která je používána v dané databázi. Aby bylo možné vzniklý otisk případně porovnávat s externími databázemi, je často vytvářeno více verzí pomocí více hashovací funkcí jako například MD5, SHA1 nebo SHA256. V případě, kdy by již vzorek existoval v databázi, není potřeba provádět další analýzu, protože její výsledek je již znám [2].

Dále je pro zjištění zda má tento malware již známou signaturu případně fingerprint prováděn sken vícero anti-virovými programy [2]. Název signatury pak může sloužit k získání dalších informací o souboru a jeho vlastnostech. Pro tuto analýzu je možné použít například nástroj VirusTotal, jenž obsahuje 7 desítek antivirů [16].

Následuje extrakce obsažených řetězců (sekvence ASCII a UNICODE) obsažených v binárním souboru. Získání těchto obsažených řetězců může velmi usnadnit další analýzu podezřelého souboru. Například pokud se malware připojuje k CnC serveru, může obsahovat doménové jméno serveru, kde se CnC nachází. Dále mohou řetězce obsahovat například názvy souborů, které malware vytváří, URL adresy, IP adresy, informace o registrech atd. [13].

Protože je extrakce obsažených řetězců často klíčovým nástrojem jak získat informace o malwaru a jeho funkcionalitě, snaží se proti ní útočnici svůj kód co nejvíce obrnit. A proto používají autoři škodlivého kódu packery (které provádí obfuskaci, šifrování a kompresi) tak, aby skryli malware před bezpečnostními experty. Tyto techniky totiž často velmi ztíží analýzu binárního souboru a oddálí tak detekci jejich malwaru.

Obsah hlavičky spustitelného souboru je dalším stěžejním bodem analýzy malwaru, obsahuje totiž důležité informace o zavedení do paměti a spuštění souboru, jeho struktuře, seznam potřebných knihoven atd. Tato analýza nám může například pomoci odhalit k čemu daný vzorek přistupuje (soubory, registry nebo třeba síť). Většina těchto operací je totiž prováděna pomocí API operačního systému a konkrétních knihoven.

Následuje porovnání vzorku s jinými a jeho klasifikace. V případě malwaru jsou klasifikovány jednotlivé vzorky podle jejich charakteristik a podobností [17]. Skupina, ve které se shlukují vzorky stejného typu, se nazývá rodina malwaru [18].

Klasifikace do malwarových rodin je používána proto, že zatímco kontrolní součet souboru je skvělým nástrojem pro určení stejného vzorku, v případě malwaru se velmi často stává, že autor škodlivého kódu změní jen malou část kódu a tím změní celou hash [13].

3.2 Dynamická analýza

Narozdíl od statické analýzy je při dynamické analýze testovaný vzorek spuštěn. Vzorek se většinou testuje v kontrolovaném prostředí, v němž se vzorek spustí a sledují se změny, které provádí v systému. Vzhledem k tomu, že malware má uzavřený kód, je na něj při analýze nahlíženo jako na blackbox. Avšak většinou se při analýze malwaru kombinují obě techniky jak dynamická tak statická analýza.

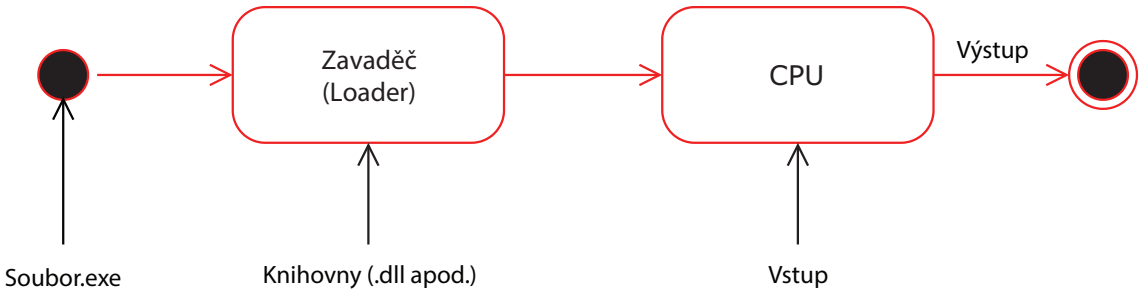
4 Formát spustitelných souborů

Spustitelný soubor je speciální druh souboru, který obsahuje instrukce pro provedení určité činnosti [19]. Například se může jednat o strojový kód nebo zdrojový kód určený pro interpret.

Oproti běžným binárním souborům není v případě spustitelných souborů potřeba žádná externí aplikace k interpretovaní těchto dat. Aby bylo možné program spustit musí být formát souboru přizpůsoben operačnímu systému [20].

Ke spuštění programu dojde tak, že program zavaděče (anglicky loader), jež je běžně součástí operačního systému, načte program do paměti a připraví potřebné knihovny pro spuštění viz obrázek č. 2.

Obrázek 2: Proces spuštění souboru



Aby ke spuštění mohlo vůbec dojít, musí operační systém podporovat formát spustitelného souboru na nízké úrovni [20]. K tomu slouží jasně definované rozhraní ABI (Application binary interface). Toto rozhraní definuje pravidla pro spolupráci strojového kódu a jádra OS. Podpora jednotného ABI na různých systémech pak umožňuje kompatibilitu zkompilované aplikace mezi různými systémy.

4.1 Spustitelné soubory v OS

Všechny operační systémy, které obsahují podporu zavedení vlastního programu, definují jeden nebo více podporovaných formátů spustitelných souborů. Tabulka č. 1 prezentuje některé spustitelné formáty a informace, na které platformě je můžeme najít.

Tabulka 1: Formáty spustitelných souborů (Zdroj dat [21])

Formát	Platforma	64-bit	Přípona
PE	Windows, ReactOS, HX DOS Extender, BeOS	Ne	.exe, .dll, .sys atd.
PE32+	Windows	Ano	.exe, .dll, .sys atd.
NE	MS-DOS, OS/2 Windows, HX DOS Extender	Ne	.exe, .dll, .fon
ELF	Unix-like, OpenVMS, BeOS, Haiku	Ano	žádná, .bin, .o, .elf, .so atd.
Mach-O	NeXTSTEP, macOS, iOS, watchOS, tvOS	Ano	žádná, .o, .dylib, .bundle

4.2 PE hlavička

Formát PE (zkr. Portable Executable) byl navržen jako nástupce předešlého formátu NE (zkr. New Executable) používaném v operačním systému MS-DOS. Tento nový formát byl poprvé uveden spolu s Windows NT. A je založen na UNIXové specifikaci COFF (zkr. Common Object File Format) [22].

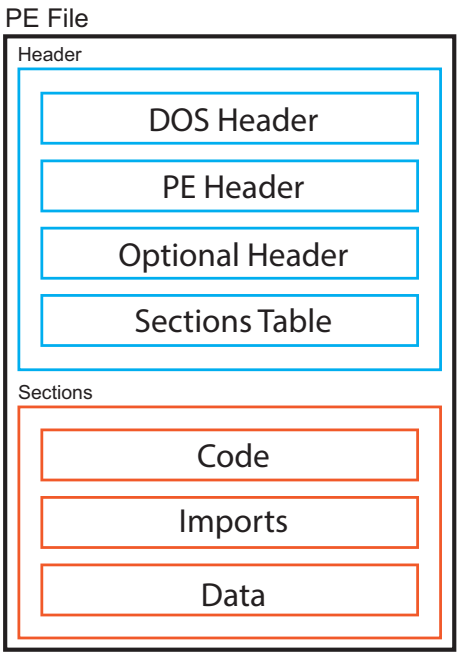
Struktura souboru Hlavička spustitelného souboru obsahuje informace o spustitelném souboru (počet sekcí, velikost sekcí atp.) a sekce pak obsahují samotné části programu (kód programu, data atp.).

Jedná se o jasně definovanou strukturu určenou pro nativně spustitelné soubory EXE a DLL knihovny [23]. Tato struktura je složena ze dvou hlavních částí a to hlavičky a sekcí [24]. Obrázek č. 3 prezentuje tuto strukturu.

Hlavička Jak už bylo zmíněno, hlavička PE souboru obsahuje celou řadu důležitých metadat pro zavedení aplikace do uživatelského prostoru. Části této hlavičky jsou dále popsány v následujícím textu.

DOS Header Prvních 64 bytů souboru ve formátu PE vždy obsahuje program pro zajištění kompatibility s OS MS-DOS. Tento program slouží k upozornění, že se nejedná o aplikaci určenou pro prostředí MS-DOS. Po jeho spuštění se vypíše hláška *This program cannot be run in DOS mode.* První hodnotou této DOS hlavičky je hodnota *e_magic* (magické číslo - viz 3.1), které identifikuje DOS mód [23]. Zároveň tato hlavička obsahuje offset (logická adresa) *e_lfanew*, jež odkazuje na začátek PE hlavičky [22].

Obrázek 3: PE formát



PE Header 4.2 Po DOS hlavičce následuje PE hlavička. Na offsetu 0x3C se nachází 4 bytový podpis identifikující soubor jako PE formát [25]. Tento podpis odpovídá hodnotě *PE\0\0* [24].

Optional Header Následuje optional header (v překladu volitelná hlavička), která je však volitelná pouze v případě objektových souborů, kde by neměla význam a pouze by zabírala místo. Pro spustitelné EXE soubory a DLL knihovny je tato část povinná [24], obsahuje totiž důležité informace pro zavaděč jako například entry point (vstupní bod programu označovaný také jako *EP*) nebo počet sekcí [22].

Sections Table Po volitelné hlavičce následuje tabulka sekcí, jež obsahuje hlavičky jednotlivých sekcí spustitelného souboru. Každá hlavička obsahuje název sekce, její velikost, umístění a charakteristiky sekce [26].

Sekce Aplikace pro OS Windows mohou využít 9 předdefinovaných sekcí jako *.text, .bss, .rdata, .data, .rsrc, .edata, .idata, .pdata a .debug* nebo si dle svých potřeb vytvořit vlastní [22].

Důležité sekce budou popsány níže.

Sekce .text Tato sekce obsahuje instrukce programu pro procesor, který je následně vykonává. Odkazuje na ni entry point v PE Header (viz) Často je to jediná sekce, ze které je program spouštěn [2].

Sekce zdrojových dat (.rsrc) Další sekcí je sekce *.rsrc*. Tato sekce obsahuje zdroje (resources) pro běh programu, které nejsou uloženy přímo v programu. Jsou to například obrázky, binární data nebo řetězce, které však mohou být přímo součástí kódu. Tyto sekce jsou často využity kupříkladu v programech, které obsahují více jazykových mutací [2].

Datové sekce (.bss, .rdata, .data) Po sekci zdrojových dat následují sekce datové. Sekce *.bss* (Block Started by Symbol) obsahuje neinicializovaná data aplikace jako statické proměnné apod. V další sekci *.rdata* jsou uložena data určená pouze pro čtení jako globální konstanty atd. [26]. Poslední typ datové sekce je pak sekce *.data*, jež obsahuje globální data, která jsou přístupná z jakékoliv části kódu v aplikaci. [2]

Sekce s exporty (.edata) Sekce *.edata* obsahuje informace k exportům jako názvy a adresy exportovaných funkcí [26].

Sekce s importy (.idata) Po sekci s exporty je nutno také zmínit sekci *.idata*, která naopak obsahuje informace k importům (adresní tabulka importu aj.) [26].

Nástroje pro analýzu Nástrojů pro analýzu spustitelných souborů existuje celá řada. V následujícím textu budou zmíněny některé z nich.

PEiD Tento nástroj je jeden z nejpoužívanějších. Je určen pro detekci známých packerů, kryptorů a kompilátorů na základě databáze signatur. V základu obsahuje databázi více než 470 signatur, je však možné přidat i své vlastní. Existují také databáze obsahující přes 3000 signatur, které je možné použít s aplikací PEiD. Detekce pomocí PEiD je ve srovnání s jinými aplikacemi pro detekci packerů velmi dobrá. Obsahuje také rozhraní připravené pro další rozšíření [27].

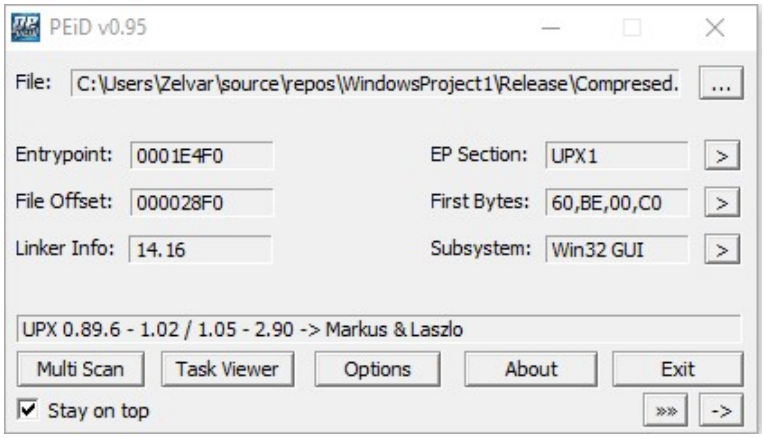
Na následujícím obrázku č. 4 je možné vidět prostředí aplikace PEiD. První část obsahuje základní informace exportované z PE hlavičky jako například, v jaké sekci se nachází entry point nebo jeho offset. Ve spodní části pak lze vidět použitý detekovaný kompilátor nebo packer (viz kapitola 6.4). V našem případě byl detekován packer UPX.

Pod základním výstupem pak nástroj nabízí různé možnosti nastavení jako například přepnutí úrovně skenování. Nabízené možnosti skenování jsou *Normal mode*, který vyhledává signatury okolo EP, *Deep mode* skenuje celou sekci. *Deep mód* detekuje až 80% souborů, které jsou chráněny před analýzou pomocí packerů. Poslední možností je *Hardcore mode*, který skenuje celý PE soubor na známé signatury [28] což však může trvat výrazně delší dobu.

Nevýhodou tohoto nástroje je, že neumí pracovat s aplikacemi určenými pro platformu x64 (PE32+). Další vývoj již neprobíhá.

Exeinfo PE Alternativou PEiD může být nástroj Exeinfo, který podporuje také PE32+. Hlavní výhodou je, že vývoj programu stále probíhá (poslední verze vyšla 22.10.2019) [29].

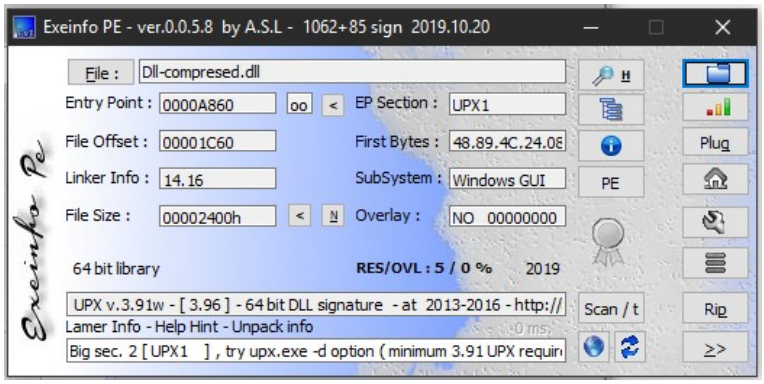
Obrázek 4: Program PEiD



Uživatelské rozhraní je velmi podobné PEiD (viz obrázek č. 5), výstup také obsahuje nápovědu s návodem k provedení unpackingu (viz kapitola 6.5).

Nevýhodou však je to, že program neumí detekovat starší způsoby ochrany (např. packery nebo protektory viz kapitola 6.4) [30].

Obrázek 5: Program Exeinfo PE

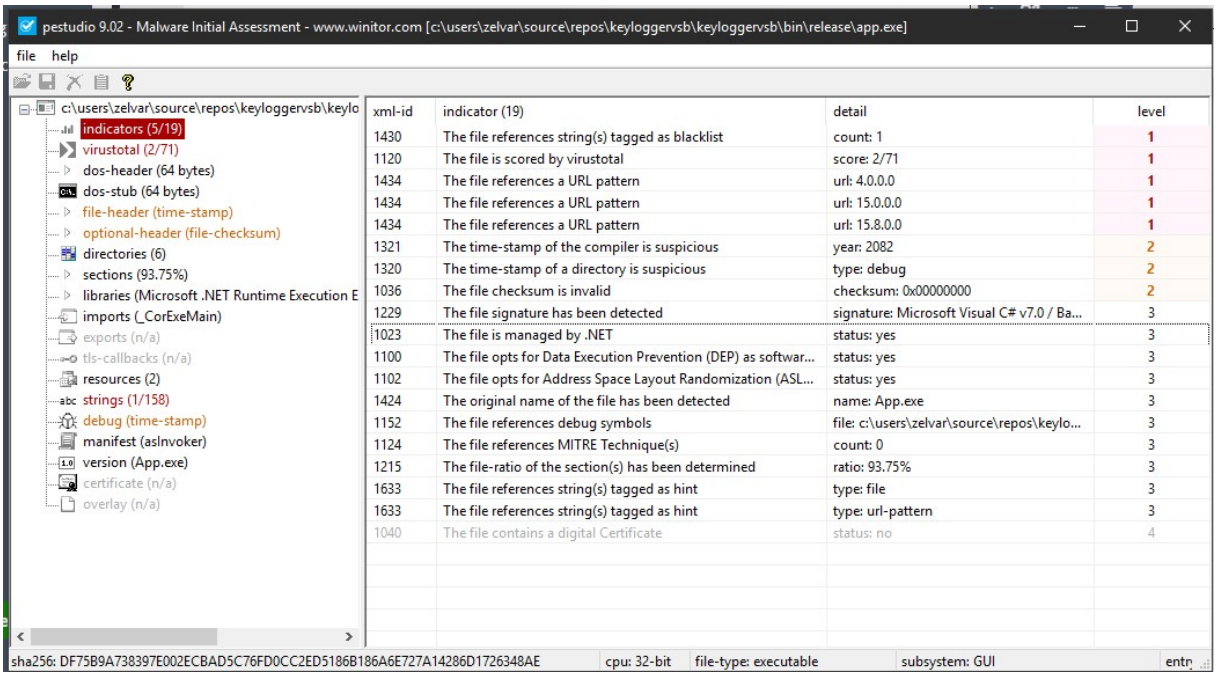


PE Studio Dalším možným nástrojem pro práci se spustitelným souborem formátu PE je PE Studio. Tento program je nabízen ve dvou variantách, jedna je zdarma a nabízí základní statickou analýzu souboru, a druhá pokročilá je nabízena za poplatek [31]. Na obrázku č. 6 je možné vidět rozhraní tohoto programu (konkrétně jeho bezplatné verze).

Základní verze programu obsahuje běžně nabízené funkce konkurenčními aplikacemi mimo jiné ale také pokročilejší funkce, které nazývá indikátory. Tyto indikátory využívají metody jako je detekce signatur, detekce URL a IP adres v kódu, vyhledávaní zakázaných řetězců ze slovníku (keyloger apod.), hledání klíčových slov mezi řetězci atp. [30].

Výstup programu je rozdělen chronologicky do několika podoken (viz obrázek č. 6) dle funkcí. Obsahuje jak již zmíněné indikátory, tak výstup z nástroje VirusTotal, seznam řetězců, seznam sekcí atd.

Obrázek 6: Nástroj PE Studio



5 Základní nástroje statické analýzy

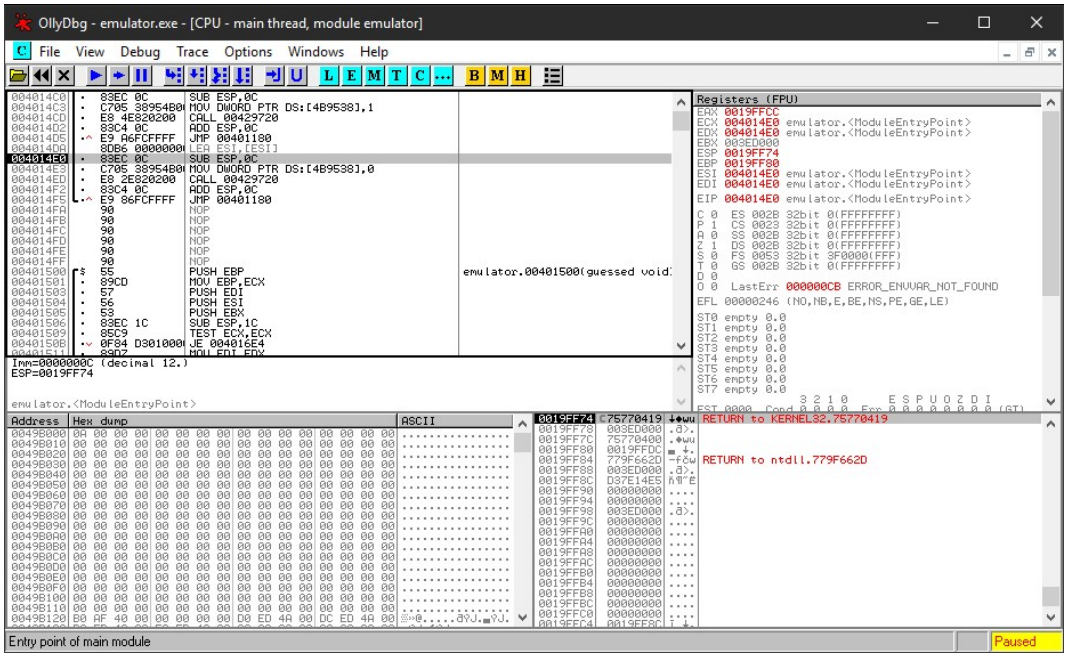
5.1 Debugger

Debugger je nástroj, který slouží primárně k ladění softwaru. Zároveň je ale také velmi důležitým nástrojem, pokud je potřeba zjistit, jak program pracuje. Umožňuje totiž krokovat kód instrukci po instrukci (v případě interpretovaného kódu pak řádek po řádku) a zároveň sledovat změny hodnot v registrech, zásobníku nebo paměti.

Další důležitou funkcí debuggeru jsou breakpointy [32]. Breakpoint lze nastavit kdekoliv v kódu a jakmile na něj program narazí, vyvolá přerušeni. Je možné tedy například zjistit stav zásobníku v každém kroku smyčky.

Debuggerů existuje celá řada. Velmi populárním je například OllyDbg viz obrázek č. 7. Tento debugger umožňuje všechny potřebné funkce pro sledování registru, volání API, konstant, stringu atd. Má přívětivé uživatelské prostředí a lze do něj vložit externí rozšíření [33]. Bohužel s ním lze debuggovat pouze 32-bitové aplikace.

Obrázek 7: Debugger OllyDbg



5.2 Disassembler

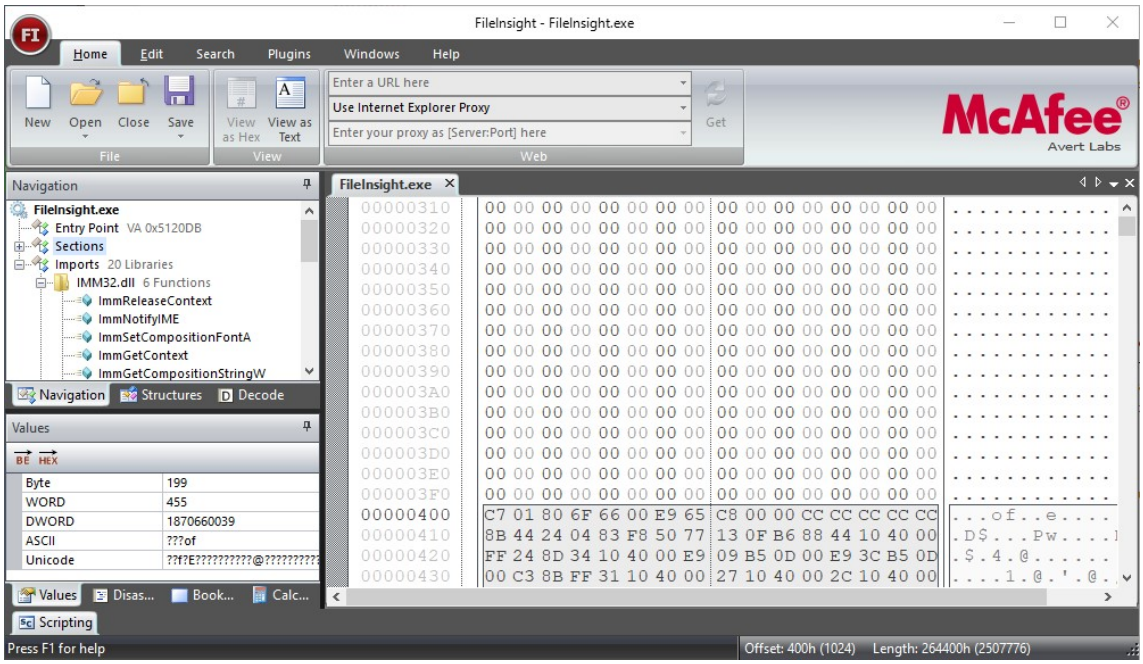
Disassembler umožňuje převod kódu zpět do programovacího jazyka nejčastěji assembleru [32]. Existují také pokročilejší disassemblery, které umí převést strojový kód do C nebo jiného jazyka. Kvalita takovýchto převodů není valná (nezískáme původní zdrojový kód) a je vhodná spíše pro lepší orientaci v konstrukcích programu. Oproti debuggeru nelze dissasembleru v jeho procesu

nijak zabránit. Proto však jsou využívány další techniky ochrany před analýzou jako obfuskace, komprese nebo šifrování.

5.3 HEX editor

HEX editor slouží ke zkoumání a editaci na úrovni bytů a bitů. Nejčastěji se souborem pracuje v šestnáctkové soustavě, proto tedy HEX editor. Některé editory také umožňuji zobrazit data jako ASCII nebo Unicode, hledat určité vzory apod. Existuje spousta HEX editorů. Vhodným může být například *FileInsight*, jehož autorem je společnost McAfee Labs. Tento editor je zachycen na obrázku č. 8. Tento HEX editor zvládne kromě běžných funkcí, jako je například editace binárních dat, také zobrazit strukturu PE souboru, dissasemblovat 32-bitové aplikace nebo dekódovat některé metody obfuskace (XOR, posun, Base64) [34].

Obrázek 8: HEX editor FileInsight



5.4 Balík sysinternals

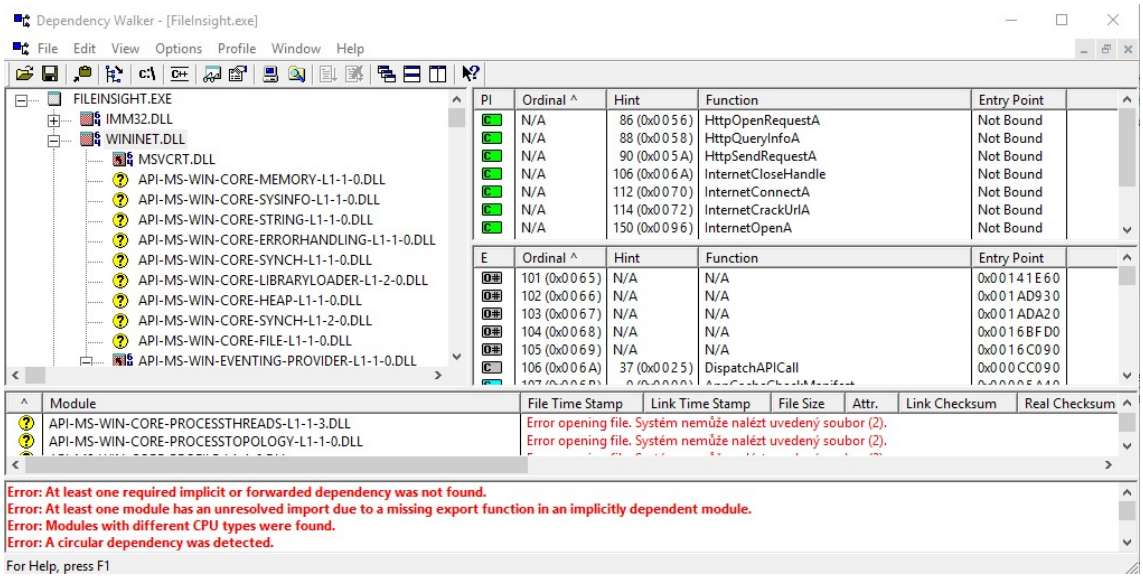
Balík programů sysinternals obsahuje nespočet utilit, které umožňují rozšířit základní funkce Windows [35]. Kolekce obsahuje například nástroje pro zjištění informací o systému, detailní správce běžících procesů apod. Tato sada je dostupná zdarma a lze stáhnout ze stránek Microsoftu.

Pro analýzu malwaru můžeme z této sady použít například aplikaci *Strings* [36], která slouží k extrakci obsažených řetězců v požadovaném souboru. V případě dynamické analýzy by bylo možné použít například *Process Explorer*.

5.5 Dependency Walker

Tento nástroj slouží k analýze závislosti zkoumaného souboru. Umožňuje analyzovat různé moduly Windows (jako .exe, .dll, .ocx, .sys atd.) [37]. A to jak 32 bitové, tak 64 bitové. Následně jsou vazby zobrazeny programem v hierarchické stromové struktuře viz obrázek č. 9. Dalším výstupem je seznam potřebných souborů včetně detailních informací o plné cestě k souboru atd.

Obrázek 9: Dependency Walker



6 Obranné mechanismy proti analýze softwaru

Postupy obrany proti analýze využívá jak běžně užívaný software pro účel ochrany duševního vlastnictví, tak i škodlivý kód k znesnadnění odhalení. Tato ochrana počítačových programů může být provedena více způsoby. Těmito způsoby se dále zabývá tato kapitola.

Ochrana duševního vlastnictví Vlastníci práv a autoři softwaru využívají různé techniky, kterými se snaží ochránit své duševní vlastnictví například algoritmy (komunikační protokol programu Skype). Často se však také jedná o multimediální obsah, který obsahuje nějakou ochranu. Příkladem mohou být počítačové hry, u kterých se vydavatelé snaží ochránit hru před tzv. crackingem, kdy se nejrůznější organizace snaží prolomit a odstranit ochranu hry.

Malware Autoři malwaru se stále snaží vymýšlet nové techniky ukrytí malwaru a znemožnění nebo alespoň ztížení jeho analýzy.

6.1 Anti-debug

Tento způsob ochrany brání ladění neboli debugovaní programu. Implementují ho často jak legitimní aplikace, které potřebuji utajit své know-how, tak i tvůrci malwaru. Existuje mnoho způsobů jak přítomnost debuggeru detekovat a případně ho blokovat. Mezi ty nejpopulárnější patří následující [2].

Windows API Microsoft Windows obsahuje ve svém API několik zpusobů jak debugger detekovat. Některé funkce byly navrženy přímo pro detekci debuggeru, některé zase pro jiné účely, ale lze je pro detekci použít.

Nejjednodušším způsobem je použít metodu *IsDebuggerPresent*. Tato metoda sleduje strukturu PEB, která obsahuje informace o aktuálním procesu, včetně pole *IsDebugged*. Pokud je toto pole nastaveno na nulu, debugger není přítomen. V opačném případě je program spuštěn v debug režimu.

Obdobným způsobem můžeme použít metodu *CheckRemoteDebuggerPresent*, která funguje na podobném principu. Rozdíl spočívá v tom, že tato funkce byla navržena ke sledování debuggu cizího (vzdáleného) procesu. Lze ji však nastavit tak, aby plnila i tento úkol.

Další možností je funkce *NtQueryInformationProcess*. Metoda opět slouží k získání informací o požadovaném procesu. Prvním parametrem funkce je ukazatel na proces. Dalším parametrem je pak možné nastavit typ získané informace.

Alternativním řešením bez sledování struktury PEB může být *OutputDebugString*. Funkce slouží k odeslání řetězce na výstup debuggeru. Pokud však debugger nebude přítomen a chybový kód bude nastaven pomocí funkce *SetLastError* bude po zavolání *OutputDebugString* na výstupu chybový kód funkce *OutputDebugString*. V případě, že by debugger přítomen byl, na výstupu bude chybový kód, který byl nastaven přes funkci *SetLastError*.

Detekce chování debuggeru Další variantou této ochrany je detekce základní funkce debuggeru a to breakpointu [38]. Breakpoint slouží analytikovi k zastavení kódu a funguje na principu vložení instrukce *INT 3* do kódu. Instrukce *INT 3* slouží k vyvolání breakpointu neboli přerušení. Malware však může skenovat sám sebe a hledat tuto instrukci v kódu. Případně vytvářet za běhu kontrolní součet nebo hash aby zjistil, zda nebylo zasaženo do kódu.

TLS callback TLS je místní paměťový prostor v rámci jednoho vlákna, kde může dané vlákno ukládat svá data [39]. Tato vlastnost může být zneužita tak, že ještě před vstupním bodem programu, kdy je inicializováno vlákno aplikace, se zavolá potřebná funkce [38].

Zneužití zranitelnosti debuggeru Stejně jako každý software i debuggery mohou obsahovat zranitelnosti. Autoři malwaru si to samozřejmě uvědomují a této skutečnosti zneužívají. Příkladem může být chyba známého a používaného debuggeru OllyDbg ve verzi 1.1 [40], která umožňovala aplikaci shodit pomocí formátovacího řetězce, zaslaného přes Windows API metodou *OutputDebugString*.

6.2 Obfuskace

Obfuskace je transformace kódu takovým způsobem, aby bylo zamezeno či alespoň znesnadněno analyzování daného kódu člověkem, přičemž funkčnost kódu setrvává [41] [42]. Je možno využít různé techniky. Tyto techniky mohou být například:

Obfuskace struktury Tato transformace se řadí mezi ty nejjednoduší. Mění strukturu zdrojového kódu programu. Přeměna je jednosměrná a již nelze získat původní zdrojový kód. Při této obfuskaci jsou odstraněny komentáře, mění se formátování (např. minifikací - odstranění nepotřebných znaků ze zdrojového kódu), je provedena změna názvů metod (metoda() = _a()) a proměnných (string heslo = string a).

Datová obfuskace Metoda datové obfuskace pracuje s maskováním datových struktur pomocí jejich přeměny na jiné sémanticky však stejné. Tento druh obfuskace se často používá pro skrytí citlivých informací. V případě malwaru se může jednat o důležité informace jako například doménové jméno či IP adresy CnC serveru, šifrovací klíče apod. Tyto transformace mohou být provedeny několika způsoby a jsou popsány níže [43].

Nahrazení statických dat Nejjednodušší způsob obfuskace statických dat je nahrazení metodou. Data jsou nahrazena kódem, který je bude dynamicky generovat. Účinnost tohoto řešení se zvyšuje s počtem volaných funkcí, jejichž volání je náhodně rozloženo do toku programu [43]. Pro zmatení reverzního inženýra je možné přidat také několik výstupů, ke kterým při normálním běhu programu nedojde.

Tuto metodu lze vidět na následující ukázce č. 1. Statická data byla nahrazena jednoduchým *switchem*, který podle čísla vstupu vrátí požadovanou hodnotu.

```
switch(int a){
    case 0:
        return "A";
    case 1:
        return "D";
    default:
        return "1337";
}
```

Výpis 1: Generování statických dat

Rozdělení proměnných Velmi účinnou technikou obfuskace proměnných je jejich rozdělení a to nejlépe za použití různých datových typů [44]. Cílem je zmást reverzního inženýra velkým množstvím použitých proměnných.

Síla a odolnost této transformace roste s počtem použitých proměnných a různorodosti jejich typů.

Následující ukázka kódu č. 2 a 3 prezentuje tuto metodu. Původní proměnnou je možné vidět v prvním výpisu č. 2 obfuskvanou proměnnou pak v ukázce č. 3. Pokud bychom definice proměnných proložili sekvencemi vykonávaného kódu docílili bychom ještě lepšího výsledku.

```
string path = "https://cnc.zelvar.cz/server.xml";
```

Výpis 2: Proměnná bez obfuskace

```
string a = "h";
var d = "cz";
a += "tt";
string c = "zelvar";
var b = "ps";
string e = "ser";
string f = "nc";

string path = a + b + ':' + new String('/', 2) + 'c' + f + '.' + c + '.' + d + '/' + e + "ver" + '.' + "xml"
    ;
```

Výpis 3: Obfuskovaná proměnná

Globalizace proměnné Máme-li funkce, například FA() a FB(), jež používají stejnou lokální proměnnou, definovanou ve svém těle a nevykonávají-li se souběžně, můžeme takovouto proměnnou deklarovat globálně a bude tak sdílena mezi oběma funkcemi [43].

Sloučení proměnných Další metodou obfuskace proměnných je jejich sloučení do jedné. Příkladem mohou být dvě 32-bitová čísla X a Y. Tato čísla mohou být sloučena do jednoho 64-bitového dle následujícího vzorce 1 [43].

$$V = 2^{32} * Y + X \tag{1}$$

Odolnost této přeměny je malá, protože deobfuskátor může odhadnout, že proměnná V se skládá ze dvou proměnných a to zkoumáním aritmetických operací.

Šifrování a kódování Další možností obfuskace dat je jejich šifrování popřípadě jejich kódování. Šifrovány mohou být buď jednotlivé části kódu nebo celé sekce spustitelného souboru [45]. Zašifrovaná data jsou následně při spuštění dešifrována [46]. Tím je zaručeno, že program funguje tak jak byl vytvořen.

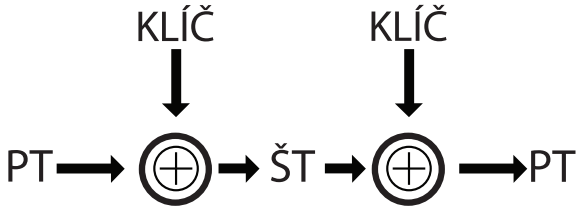
Často je možné se setkat s šifrováním pomocí bitové funkce *XOR* nebo kódováním do *Base64*. Takovéto zakrytí dat může oddálit odhalení skutečného obsahu, protože nemusí být na první pohled patrné, že jsou data šifrována nebo kódována.

Kódování *Base64* bylo původně navrženo pro přenos příloh v emailové komunikaci a je součásti standardu MIME [2]. Kódování *Base64* umožňuje převést binární data na ASCII řetězec a, jak značí číslo 64 v názvu kódování, používá se právě 64 (resp. 65) znaků US-ASCII. Znaky užívané kódováním jsou a-z, A-Z, 0-9, +, / a = pro zarovnání délky [47].

Implementace do vlastního programu je velmi jednoduchá a většina moderních programovacích jazyků už toto kódování umožňuje. Útočnici (autoři malwaru) tak často využívají toto kódování ke skrytí závadného kódu nebo dat.

Šifrování pomocí exkluzivní disjunkce (*XOR*) je velmi oblíbeným způsobem šifrováním mezi tvůrci malwaru [2], protože se jedná o oboustrannou funkci, není potřeba implementovat různé algoritmy pro šifrování a dešifrování 10. Klíčem je vždy proud bitů [48].

Obrázek 10: De/šifrování pomocí XOR



Obfuskace běhu programu Tato transformace je využívána za účelem maskování posloupnosti kódu. Provádí se například záměnou pořadí prováděných instrukcí pomocí podmíněných skoků, vloženým nahodilým mrtvým kódem atd. [41].

Změna pořadí prováděného kódu Tento typ obfuskace spočívá v tom, že jsou provedeny skoky mezi jednotlivými částmi programu, které jsou náhodně rozmístěny. Přestože probí-

hají skoky, chování počítačového programu se nezmění. Tato metoda je v podstatě jednoduchá nicméně v delším kódu komplikuje analýzu pořadí prováděných operací [41].

Princip prezentuje následující výstup kódu č. 4 a 5. Kód byl rozdělen do několika části (*.start, .end a .continue*) následně byl náhodně rozmísten. Podmíněné skoky (*JMP*) pak zajistí aby došlo k vykonání kódu ve správném pořadí.

```
arrayMax:
    enter 0,0

    mov eax, [ edx ]
    mov edx, [ ebp + 8 ]
    mov ecx, [ ebp + 12 ]

.back:
    cmp eax, [ edx + ecx * 4 - 4 ]
    jg .skip
    mov eax, [ edx + ecx * 4 - 4 ]
.skip:
    loop .back

    leave
    ret
```

Výpis 4: Původní kód funkce bez obfuskace

```
arrayMax:
    enter 0,0

    jmp .start
.continue:
    mov ecx, [ ebp + 12 ]

.back:
    cmp eax, [ edx + ecx * 4 - 4 ]
    jg .skip
    mov eax, [ edx + ecx * 4 - 4 ]
.skip:
    loop .back
    jmp .end

.start:
    mov eax, [ edx ]
    mov edx, [ ebp + 8 ]
    jmp .continue

.end:
    leave
    ret
```

Výpis 5: Obfuskovaný kód funkce

Vložení nahodilého kódu Tato technika obfuskace je založena na vložení mrtvého nebo nepodstatného kódu do programu. Využívají ji především útočnici, protože pomocí této metody lze vytvořit novou verzi programu, která se chová stejně. Účel spočívá především ve zmatení detekce antiviry, protože vložený kód změní signaturu programu (známé signatury jsou využívány pro rychlou detekci malwaru) [41].

Ukázka následujících výpisu č. 6 a 7 prezentuje rozdíl hlavně v délce kódu. Do kódu bylo vloženo několik skoků pomocí *JMP* a instrukcí *NOP*, která nevykonává žádnou operaci.

```
sumArray:
    enter  0,0
    mov eax, 0
    mov edx, [ ebp + 8 ]
    mov ecx, [ ebp + 12 ]
    jcxz  .skip
.back:
    add eax, [ edx + ecx * 4 - 4 ]
    loop  .back
.skip:

    leave
    ret
```

Výpis 6: Původní kód bez obfuskace vloženého kódu

```
sumArray:
    enter  0,0
    nop
    jmp .skipcode
    cmp eax, [ edx + ecx * 4 - 4 ]
    jg  .skip
    mov eax, [ edx + ecx * 4 - 4 ]
.skipcode:
   nop
    mov eax, 0
    nop
    mov edx, [ ebp + 8 ]
    mov ecx, [ ebp + 12 ]
    jcxz  .skip
.back:
    add eax, [ edx + ecx * 4 - 4 ]
    loop  .back
.skip:

    leave
    ret
```

Výpis 7: Kód funkce po obfuskaci

Nahrazení ekvivalentem Díky nahrazení části kódu ekvivalentem lze provádět stejnou funkcionalitu programu různými způsoby. Tím je možné dosáhnout změny signatury, což způsobí vyšší náročnost detekce malwaru, takto vznikají nové verze u nichž je potřeba uchovat signaturu každé nové jedinečné verze [41].

Následující výpisy kódu č. 8 a 9 prezentují tuto metodu. Kód se liší ve dvou podstatných instrukcích, jež vykonávají ekvivalentní funkci a to instrukce *test* nahrazena ekvivalentem *cmp* a instrukce *inc* nahrazena *add*.

```
strlen:
    enter 0,0
    mov edx, [ ebp + 8 ]
    mov eax, 0
.back:
    test byte [ edx + eax ], 0xff
    jz .exit
    inc eax
    jmp .back
.exit:
    leave
    ret
```

Výpis 8: Kód před nahrazením

```
strlen:
    enter 0,0
    mov edx, [ ebp + 8 ]
    mov eax, 0
.back:
    cmp byte [ edx + eax ], 0
    jz .exit
    add eax, 1
    jmp .back
.exit:
    leave
    ret
```

Výpis 9: Kód po nahrazení

6.3 Komprese

Cílem komprese spustitelných souborů je zmenšit celkovou velikost kódu a sekcí s daty. Původní spustitelný soubor se nahradí souborem novým. Tento nový soubor obsahuje program pro dekompresi a původní spustitelný soubor jež je komprimovaný. Pří spuštění se původní soubor nejdřív dekomprimuje do paměti a následně se spustí [49].

Běžné formáty spustitelných souborů, jako PE, ELF atp. v základu kompresi nepodporují. Proto v době, kdy byl nedostatek paměti a malá velikost spustitelných souborů byla tedy nutností, začala tak vznikat různá řešení pro kompresi spustitelných souborů.

V současnosti se však komprese využívá spíše z důvodu možnosti skrýt obsah spustitelného souboru bez jeho dekomprese.

6.4 Packery

Při použití packingu dochází k zabalení původního programu nebo jeho části a to pomocí packeru nad binárními daty. Může docházet k zašifrování nebo kompresi. Packer nahradí původní program nebo jeho část tzv. unpackerem. Při spuštění kódu se nejprve provede tzv. unpacking do paměti a následně se kód spustí [50]. Tvůrci malwaru využívají tuto techniku packování velmi často, za účelem ztížení a časové prodloužení analýzy kódu reverzním inženýrem [51].

Při svém procesu mohou zároveň provádět kompresi, obfuskaci, šifrování nebo přidat jinou funkcionalitu pro ztížení reverzní analýzy [52]. Druhy packerů si popíšeme v následujícím textu, součástí bude porovnání několika packerů viz tabulka č. 2. Některé packery, jež jsou porovnávaný jsou veřejně dostupné a mají také otevřený zdrojový kód (UPX, Obfuscator, ConfuserEX). Otevřený kód může nahrávat reverzním analytikům, protože snáze zjistí jak packer funguje a mohou tak tedy kód případně snáze unpackovat. Existují však také komerční packery, které otevřeným kódem nedisponují a je těžší je analyzovat.

Rozšiřující

Kompresory Tento druh packeru slouží primárně k snížení velikosti spustitelného souboru. Před spuštěním se provede dekomprese do paměti a soubor se spustí [50].

Protektor Cílem protektoru je ztížit analýzu kódu různými metodami jako anti-debug, anti-vm apod. a ochránit tak kód vůči reverzní analýze [53].

Kryptor Dalším druhem packeru je kryptor, jež provádí šifrování originálního souboru. Před spuštěním je nejprve dešifrován a následně spuštěn [53].

Bundler Tato metoda packování umožňuje zabalit do jednoho souboru všechny potřebné soubory. Program se tedy na první pohled tváří, že neobsahuje žádné externí knihovny apod. [50].

Transformující

Virtualizátor Virtualizátory převádí původní spustitelný kód do jazyku virtuálního stroje, který následně provádí vestavěný virtuální stroj [50].

Mutátor Převádí instrukce na alternativní (v rámci stejné platformy). Využívá se oligomorfismu [50].

Tabulka 2: Výběr některých packerů

Název	Licence	x86-64	Komprese	Obfuskace	Šifrování	Jiná
UPX	GPL	✓	✓			
ASPack	Proprietární	x86	✓			
ASProtect	Proprietární	✓	✓		✓	✓
Enigma Virtual Box	Proprietární	✓	✓	✓		✓
Obfuscar	MIT	.NET	✓	✓		
ConfuserEx	MIT	.NET	✓	✓	✓	✓

6.5 Unpacking

Unpacking je proces, při kterém dochází k obnově původních zdrojových dat (kódu) z programu, jež byl zabalen jedním nebo vícero packery [51]. Existují tří různé způsoby jak získat původní kód [54]. A to buď pomocí ručního, statického nebo dynamický unpacking [55].

Ruční unpacking Při ručním unpackingu reverzní analytik zkoumá jednotlivé vrstvy algoritmu, jímž byl kód šifrován, komprimován apod. A ručně se snaží o obnovu původních dat. Tato technika je však velmi časově náročná, vyžaduje hlubší znalosti nižších vrstev OS a také znalost assembleru [54].

Statický unpacking Metoda statického unpackingu je vlastně způsob jak zautomatizovat unpacking známých packerů jako například kompresor UPX. Jde o rutinní operace, které provádí dekompresi, dešifrování apod. A umožňují tak snadno získat původní kód. Autoři malwaru však mohou mírně upravit packer nebo použít svůj vlastní a původní unpacker již nebude funkční [54].

Tuto metodu také používají antivirové společnosti ve svém softwaru, aby urychlili detekci nových neznámých vzorků malwaru [54].

Dynamický unpacking Zatímco statický unpacking se snaží nahradit proces packeru, jež se stará o rozbalení a spuštění původní aplikace. Dynamický unpacker nechá rozbalení na původním programu. Unpacker nejdříve spustí zabalený program, nechá jej až se rozbalí do paměti. A pak se snaží získat rozbalený kód z paměti a uložit do souboru [55].

6.6 Anti-VM

Autoři malwaru jsou si plně vědomi využití izolovaného virtuálního prostředí při dynamické analýze. Proto do svých programů implementují techniky, jež detekují takové virtuální prostředí. V případě, že malware toto prostředí detekuje, může například deaktivovat svou funkčnost. V případě statické analýzy, lze některé z těchto postupů odhalit.

Tvůrci škodlivého kódu, jež takovouto obranu implementují do svého programu, využívají znalostí o izolovaném prostředí. Tyto postupy jsou popsány níže [1]. (Při testování těchto funkcionalit byl použit VMware verze 15.5.1 a VirtualBox ve verzi 6.0.10)

Instrukce CPU Prvním způsobem jak detekovat virtuální prostředí je použití instrukce CPUID, která slouží k zjištění informaci o daném procesoru.

Následující ukázka č. 10 prezentuje instrukci CPUID pro zjištění, zda se program nachází ve virtuálním prostředí. Nejprve je nastaven registr EAX = 1 a následně je vykonána instrukce CPUID. Ta při nastaveném EAX na 1 vrátí do několika registrů (EAX-EDX) informace o procesoru. [56] V tomto případě bude je předmětem zájmu registr ECX, který na 31. bitu obsahuje informaci, zda se jedná o hostitele nebo prostředí VM. Pokud bude tento bit nastaven na 0 jedná se o fyzický stroj, v opačném případě půjde o hosta [1].

```
testVM:
    enter 0,0
     mov eax, 1
     cpuid
     bt ecx, 31
     jb .vm
     mov eax, 0
     jmp .end
.vm:
     mov eax, 1
.end:
     leave
     ret
```

Výpis 10: Instrukce CPUID - Převzato z [1]

Obdobně lze využít instrukci CPUID viz ukázka č. 10 k získání názvu hypervizoru virtualizačního softwaru. Postup je obdobný je potřeba nastavit registr EAX na hodnotu 40000000 a následně provést instrukci CPUID. Po provedení instrukce s tímto parametrem je následně nastavena hodnota registrů EAX, ECX a EDX. Získané názvy jsou uvedeny v tabulce č. 3.

Tabulka 3: Názvy získaných hypervizorů pomocí funkce CPUID

Hypervizor	Obsah registrů
VirtualBox	VBoxVbox
VMware	VMwareVMware
Hyper-V	Microsoft HV

V případě VMware, může být použita také detekce díky tzv. VMWare Magic Number [2] (viz ukázka č. 11) ((*0x56 0x4D 0x58 0x68*) = řetězec VMXh dle hodnot v ASCII tabulce). V tomto případě se využívá specifický I/O port. Nejdříve jsou nastaveny registry EAX na hodnotu VMXh a číslo portu v registru EDX (*0x56 0x58* = VX řetězec dle ASCII hodnot). Následně se zavolá instrukce IN pro čtení z tohoto portu. Pokud po této instrukci dojde k přepsání registru EBX tzv. kouzelným číslem, dojde k úspěšnému připojení k tomuto portu a proniknutí dovnitř VMWare.

```
VMwareTest:
    enter 0,0
     mov eax, 0x564D5868 ; VMXh
     mov edx, 0x5658 ; VX (port)
     in eax, dx
     cmp ebx, 0x564D5868 ; VMXh
     setz ecx
     mov eax, ecx
     ret
     leave
```

Výpis 11: Anti VMware - Převzato z [2]

MAC adresy Také MAC adresy mohou sloužit k identifikaci virtualizace [1]. Vodítkem pro autory malwaru může být seznam registrovaných bloků spravovaný organizací IEEE [57]. Tento seznam obsahuje adresy a výrobce, který si je registroval. Viz tabulka č. 4.

Tabulka 4: Některé MAC adresy, které se používají ve virtuálním prostředí - převzato z [1]

Výrobce	Blok
VMware	00:05:69
VMware	00:0C:29
VirtualBox	08:00:27
VirtualBox	00:21:F6
Privátní rozsah	00:00:6C
Privátní rozsah	78:F9:44

Ovladače hardware Protože většina součástí hardwaru je virtualizována, jsou potřeba specifické ovladače. Tyto ovladače je možné nalézt například pomocí jednoduchého Powershell příkazu, který je možné vidět na následujícím výpisu č. 12.

```
gwmi Win32_SystemDriver | Where-Object {$_.DisplayName -like "*VirtualBox*"} | select DisplayName
```

Výpis 12: Powershell kód pro získání instalovaných ovladačů VirtualBoxu

Tento skript umožňuje získat seznam názvů nainstalovaných ovladačů obsahujících klíčové slovo VirtualBox. Na následujícím výstupu č. 13 a 14 konzole můžeme vidět, že názvy ovladačů hosta jsou dostatečně jedinečné a tedy rozpoznatelné od hostitelských ovladačů.

```
DisplayName
-----------
VirtualBox Service
VirtualBox NDIS 6.0 Miniport Service
VirtualBox NDIS6 Bridged Networking Service
VirtualBox USB
VirtualBox USB Monitor Service
```

Výpis 13: Seznam hostitelských ovladačů VirtualBoxu

```
DisplayName
-----------
VirtualBox Guest Driver
VirtualBox Guest Mouse Service
VirtualBox Shared Folders
```

Výpis 14: Seznam ovladačů VBox hosta

Běžící procesy a služby Obdobně jako ovladače je možné na virtualizovaném stroji nalézt také specifické procesy případně služby, které indikují, že se jedná o virtuální prostředí. Pro získání seznamu běžících procesů a služeb lze použít následující dva příkazy viz výpis č. 15.

```
Get-Process | Where-Object {$_.ProcessName -like '*vm*'} | select ProcessName
Get-Service | Where-Object {$_.Name -like '*VBox*'} | select DisplayName
```

Výpis 15: Powershell kód pro získání běžících procesů a služeb

Výstupem je opět možné ověřit, že běžící procesy hostitele a hosta se liší. První tabulka na výstupu č. 16 obsahuje seznam služeb, druhá (viz výstup č. 17) pak seznam běžících procesů.

```
DisplayName
-----------
VirtualBox system service

ProcessName
-----------
VirtualBoxVM
VirtualBoxVM
VirtualBoxVM
```

Výpis 16: Výstup konzole hostitele

```
DisplayName
-----------
VirtualBox Guest Additions Service

ProcessName
-----------
```

Výpis 17: Výstup konzole hosta

Registry V neposlední řadě mohou být vodítkem pro autory malwaru také registry na OS Windows, kde je možné nalézt záznamy o existenci nástrojů virtuálního prostředí případně další specifické záznamy [1].

Některé z těchto záznamů (převzato z [58]) lze nalézt v následujících cestách:

- VirtualBox - HKLM:/HARDWARE/ACPI/DSDT/VBOX___;
- VirtualBox - HKLM:/HARDWARE/ACPI/FADT/VBOX___;
- VirtualBox - HKLM:/SOFTWARE/Oracle/VirtualBox Guest Additions;
- VMware - HKLM:/Software/VMware.

7 Současný stav poznání

Tato kapitola se zabývá přehledem vybraných prací, které reprezentují aktuální přístupy ke statické a dynamické analýze škodlivého kódu a jejich výhodami a nedostatky.

7.1 Statická a dynamická analýza malware pomocí strojového učení

Autoři [59] zde získávají charakteristiky, a to pomocí jak statické, tak dynamické analýzy. Statická analýza je zahájena extrakcí hlavičky a jednotlivých sekcí PE file pomocí knihovny v pythonu bez spuštění v kontrolovaném prostředí. Jsou analyzovány jednotlivé sekce. Dále je také zahájena analýza dynamická, kde již dochází ke spuštění malwaru v kontrolovaném prostředí (konkrétně je použito Cuckoo sandbox), který zaznamenává chování malwaru a po ukončení analýzy vrátí do původního stavu. Poté jsou pomocí různých metod strojového učení zkoumány použité API calls, IP adresy a DNS fronty atd. Celkem je pomocí kombinace statické a dynamické analýzy dosaženo výsledku v podobě více než 92 různých charakteristik. Podle autorů je problémem to, že malwary snadno identifikují kontrolované prostředí a mohou tomu tedy přizpůsobit i své chování. V porovnání s dynamickou analýzou se jeví statická analýza jako účinnější. Nicméně i tato je limitována, a to převážně velikostí malwaru a možností jeho obfuskace. Kombinace statické i dynamické analýzy je tedy vhodná, do budoucna je však potřeba zajistit odstranění nedostatků jednotlivých postupů.

7.2 Klasifikace malwaru na základě charakteristik získaných pomocí statické analýzy

Parametry získané statickou analýzou jsou autory [60] pomocí metod strojového učení tříděny do jednotlivých kategorií. Jako zdroj dat zvolili autoři databázi VirusShare. Pro klasifikaci malwaru byl vybrán systém, který využívá Kaspersky scan. Nejprve byly vzorky prozkoumány pomocí Exeinfo PE a Kapersky scanu, tím bylo získáno rozdělení do jednotlivých skupin. Poté byly pomocí statické analýzy získány hexadecimální charakteristiky bytekódu, assembleru a také struktury PE filů. Ty byly dále testovány. Nakonec byly vybrány charakteristiky, na kterých byly aplikovány různé algoritmy strojového učení (Scikit-learn knihovny strojového učení zahrnující SVM, rozhodovací strom, náhodný les atd.).

7.3 Statická analýza malwaru v systému android: techniky, limity a přetrvávající výzvy

V tomto článku si autoři [61] zvolili za cíl vyhledat nedostatky statické analýzy prezentované v již existující literatuře a tyto poté diskutovat (jsou zmíněny různé druhy obfuskace, šifrování řetězce, vložení nadbytečného kódu, apod., využívání dex či jar souborů, přidávání škodlivého kódu k ověřeným souborům) dále také vysvětlit čtyři fáze detekce malwaru (fáze předcházející zpracování, fáze extrakce charakteristik, fáze výběru charakteristik a fáze detekce), rozebrat

charakteristiky, které jsou využívány při statické analýze, a nakonec prezentovat srovnání mezi využitím komerčních antivirů a nástroje vyvinutého pro odhalení obfuskovaného malwaru pomocí statické analýzy. Dochází k závěru, že je stále potřeba zaměřit se na vývoj nových nástrojů, které by pomohly s odhalením malwaru využívajícího pokročilé metody obfuskace.

7.4 Limity statické analýzy pro detekci malwaru

Tento článek [62] se zabývá identifikací limitů statické analýzy za účelem detekce škodlivého kódu. Hlavním způsobem zamezení nebo znesnadnění statické analýzy je dle autorů obfuskace. Která je provedena pomocí přepisovacích nástrojů (určených pro Windows či Linux), které nemají dostupné zdrojové kódy ani jinou dokumentaci. Nechybí zde výsledky, které ukazují, že detektory malware zohledňující sémantiku mohou být snadno obejity a zároveň je předvedeno, že binární změny provedené autory škodlivého kódu jsou robustní.

Obfuskace použita autory škodlivého kódu, má za cíl nahradit některé části původního kódu, takovými, které jsou sémanticky souhlasné a zároveň znesnadňují statickou analýzu, ale nemění programu [62].

Zároveň bylo také ověřeno na reálných malwarech, že změny zdrojového kódu pomocí obfuskace opravdu znemožňují odhalení malware pomocí statických metod. Konkrétně autoři využili tři červy (MyDoom.A, MyDoom.AF, Klez), které upravili popsanými metodami a ověřili, že fungují i nadále správně, a dále čtyř běžných antivirů (McAfee Anti-Virus, Kaspersky Anti-Virus, AntiVir Personal Edition a Ikarus Virus Utilities) a jednoho pokročilého nástroje analýzy software (IDA Pro) [62].

Po otestování tři obfuskovaných červů jednotlivými antiviry se ukázalo, že McAfee identifikoval tyto červy na základě jejich podpisu. Poté co byl podpis v dané sekci upraven, k odhalení už nedošlo. Oproti tomu AntiVir kontroluje i další parametr. Pokud byly změněny oba tyto parametry nedošlo k identifikaci. Nástroj statické analýzy nejprve rozebere binární kód pomocí IDA Pro a následně provede modelovou analýzu. (zjistí, zda se nevolá nějaká sekvence funkcionalit Windows API, například pro kopírování do jiné části systému pomocí GetModuleFIleNameA a CopyFileA). Identifikace na základě sémantické sekvence kódu je odolná vůči změně uspořádání. I přes to po obfuskaci červu nebyl nástroj schopen identifikovat žádného z nich [62]. (volání knihoven již nebylo identifikováno)

Autoři předpokládají možná řešení. Jednim z nich je možnost označit programy, které jsou obfuskovaný jako podezřelé, to se ovšem pojí s řadou falešně pozitivních výsledků. Slibnějším přístupem, dle autorů, je dynamická analýza, kdy se obfuskace stává bezpředmětnou.

7.5 Analýza výkonu strojového učení a algoritmu pro rozpoznávání schémat pro klasifikaci malwaru

Autoři [63] pomocí imagery jsou kódy vizualizovány jako binární schémata, ta jsou dále upravena jako 2D matice a následně transformována do formy obrázku (velikosti 256x16). Ukazuje

se, že u malwaru ze stejné rodiny mají obrázky podobnou texturu. Klasifikace malwaru je prováděna pomocí PCA (Principle Component Analysis nebo také analýza hlavních komponent), kNN, ANN (Artificial Neural Networks tedy umělých neuronových sítí) a SVM (Support Vector Machine – metoda podpůrných vektorů). Nakonec jsou srovnány výstupy jednotlivých metod klasifikace. Jako nejlepší vychází z porovnání kNN. Dále je také zjištěno, že analýza obrázku namísto kódu zkracuje dobu výpočtu bez toho, aby došlo ke snížení výkonu.

7.6 Technologické sítě a šíření počítačových virů

V tomto textu se autoři [64] zabývají pochopením struktury, ve které se šíří malware, protože podobně jako u lidské nákazy je pro kontrolu šíření infekce stěžejní pochopit, v jaké struktuře se nákaza šíří. Kontrola šíření infekce pomocí náhodné vakcinace je pro zařízení připojená například k internetu či world wide webu nedostatečná, protože se jedná o takzvané sítě bez měřítka. Viry navíc mohou cíleně strategie obcházet, například se mohou začít šířit v sítích s měřítkem namísto sítí bez měřítka. Autoři dále porovnávají čtyři typy sítí – síť, která využívá propojení přes IP adresu, síť, ve které může administrátor číst a přepisovat data na disku, síť, která propojuje uživatele pomocí emailových adres v jejich adresáři a síť, kde je propojení uživatelů dáno nedávnou výměnou emailu. Tyto typy sítí jsou mezi sebou porovnány a na základě jejich topologie a distribuce jsou navrženy strategie vakcinace. Ku příkladu je zmíněn throttling, který umožňuje omezit množství nových připojení a tím zabraňuje rychlému šíření malwaru v síti. To umožňuje nový malware prozkoumat a vyvinout obranné mechanismy (například aktualizovat software).

7.7 Analýza a klasifikace malwaru: průzkum

V tomto textu [65] je uveden přehled technik a přístupů k analýze a klasifikaci škodlivého softwaru. Jsou zde zmíněny také nástroje, které je možné využít jak pro statickou (IDA Pro, OllyDbg, LordPE nebo OllyDump), tak i pro dynamickou analýzu (Process Monitor, Capture BAT, Proces Explorer, Process Hackrreplace, Wireshark, Regshot, Norman Sandbox, CW Sandbox, Anubis, TTAnalyzer, Ether a ThreatExpert). Autoři také uvádí různé přístupy strojového učení (Association Rule, Support Vector Machine, Decision Tree, Random Forest, Naive Bayes, Clustering).

8 Experimentální část

V rámci této práce byla implementována aplikace, jež provádí statickou analýzu požadovaného souboru se zaměřením na spustitelný formát *Portable Executable* (viz kapitola 4.2), a následně se pomocí strojového učení snaží testovaný vzorek klasifikovat.

Pro vývoj byl zvolena platforma .NET framework. Motivací výběru tohoto frameworku, byla možnost vývoje jak desktopové, tak webové aplikace.

8.1 Použité nástroje a knihovny

Při vývoji této aplikace bylo pro usnadnění jejího vývoje použito několik knihoven. Veškeré knihovny byly přidány do projektu pomocí správce balíčku službu určenou pro platformu .NET, NuGet. Tyto knihovny jsou popsány níže.

PeNet PeNet je .NET knihovna, jež je určená pro parsování (což je syntaktická analýza) spustitelných souborů ve formátu PE (viz kapitola 4.2) [66]. Tato knihovna je implementována ve standardizovaném formátu .NET Standard (viz 8.2). Je tady možné tuto knihovnu použít jak pro aplikace na Windows, tak například pro analýzu souboru v prostředí OS s jádrem Linuxu.

Tato knihovna umožňuje například extrakci jednotlivých části PE formátu, vytvoření otisku jednotlivých části jako jsou např. importy atd. A to bez jakékoliv závislosti na Windows API [66].

Knihovna má otevřený zdrojový kód a je dostupná pod licencí Apache 2.0. Může tak být použita v proprietárním softwaru [66].

VirusTotalNET Pro práci s webovou aplikací VirusTotal, byla využita knihovna VirusTotalNET. Tato knihovna umožňuje spolupráci s API ve verzi 2 [67].

VirusTotal umožňuje provedení analýzy jakéhokoliv souboru nebo url adresy. Služba obsahuje přes 70 antivirů, je tak skvělým nástrojem pro prvotní kroky analýzy [68].

ML.NET Tato knihovna přidává nativní podporu strojového učení do frameworku *.NET*, což umožňuje rozšířit jakoukoliv aplikaci na platformě *.NET* o predikci stavu, detekování anomálií nebo třeba více třídní klasifikaci [69].

YARA YARA je nástroj určený pro identifikaci a klasifikaci vzorků primárně malwaru. Pomocí tohoto nástroje můžeme rozdělovat jednotlivé vzorky do skupin, a to na základě textových nebo binárních vzorů [70].

Yara pracuje na principu vyhledávaní těchto vzorů dle předepsaných pravidel. Tato pravidla mohou obsahovat sadu řetězců nebo binárních dat. Pravidla si buď můžeme vytvořit sami nebo použít jednu z dostupných otevřených databází. Poslední alternativou pak může být použití nástroje pro vygenerování těchto pravidel *YarGen* [71].

Následující příklad, prezentuje ukázkové pravidlo (viz výpis zdrojového kódu č. 18). První řádek definuje dané pravidlo jako *Example* následně je pravidlo rozděleno do několika částí. V našem případě *meta*, *strings* a *condition*. První část *meta* obsahuje metadata o tomto pravidlu, jako jsou například autor, popis apod. Část *strings* obsahuje řetězce, které by měl testovaný vzorek obsahovat. V této části můžeme použít řetězec vyjádřený pomocí šestnáctkové soustavy, běžně vyjádřeného řetězce nebo regulární výraz. V poslední části se pak nachází podmínka, kterou musí testovaný vzorek splnit, aby pravidlo bylo vyhodnoceno pozitivně. V našem případě musí vzorek obsahovat buď řetězec z proměnné *$text1* nebo *$text2*.

```
rule Example {
   meta:
      description = "DP ukazka pravidla"
      author = "Jan Hložek"
   strings:
      $text1 = "hello"  //hello string
      $text2 = {68 69} //hi string
   condition:
      $text1 or $text2
}
```

Výpis 18: Ukázka pravidla YARA

Použití v .NET Pro použití nástroje YARA na platformě .NET bylo využito knihovny *libyara.NET* vyvíjené společností *Microsoft*, jež zajišťuje zjednodušené volání API pomocí adaptéru (angl. wrapper) knihovny libyara na platformě .NET [72].

MIME Protože se nelze spoléhat pouze na příponu souboru. Bylo pro identifikaci typu souboru, využito standardu *MIME* (zkr. Multipurpose Internet Mail Extensions). Tento standard byl původně navržen pro přílohy v emailové komunikaci. A obsahuje mimo jiné přesnou specifikaci názvu jednotlivých formátu.

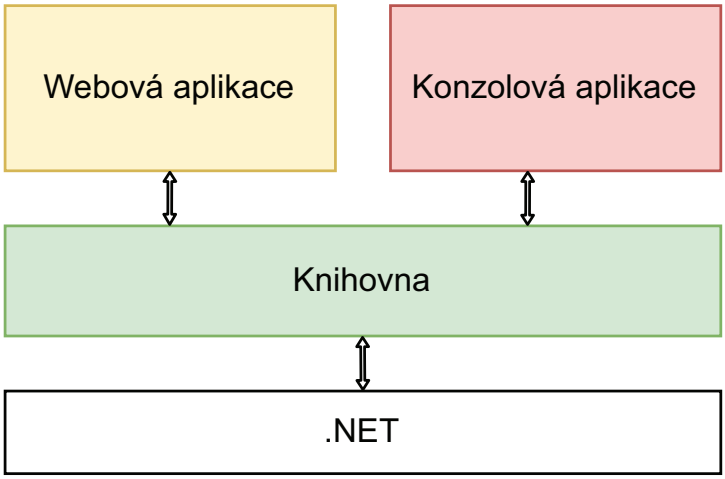
Pro tento účel byla v .NET využita knihovna se stejným názvem *MIME* [73]. Knihovna slouží jako adaptér pro knihovnu *libmagic*, jež obsahuje databázi tzv. magických čísel (angl. magic numbers) viz kapitola 3.1. Ty pak slouží k detekci formátu souboru [73]. Výstupem je pak standardizovaný název formátu souboru např. *application/x-dosexec*, jež odpovídá spustitelnému souboru pro operační systém Windows nebo DOS.

8.2 Popis aplikace

Vznikající program statické analýzy byl od začátku koncipován jako jednoduchá, modulárně rozšiřitelná platforma, která bude umožňovat provedení přehledného výstupu. A to včetně případného automatického vyhodnocení testovaného vzorku pomocí knihovny ML.NET (viz kapitola 8.1).

Architektura aplikace byla rozdělena do dvou vrstev viz obrázek č. 11. První vrstva zajišťuje veškeré prováděné kroky statické analýzy. Jako analýzu řetězců, přípravu výstupů apod. Tato část byla vyvíjena ve specifikaci .NET standard, jež zajišťuje jednotnost chování programu na různých platformách [74]. Druhá vrstva pak zajišťuje samotný výstup aplikace, v našem případě byla aplikace vyhotovena ve dvou provedeních. A to jako webová aplikace, tak jednoduchá konzolová aplikace.

Obrázek 11: Jednotlivé vrstvy aplikace



Průběh statické analýzy byl rozdělen do několika dílčích kroků. Jednotlivé kroky pak zabezpečují samotné moduly programu, kterých bylo v této práci vytvořeno celkem sedm.

Každý konkrétní modul provádí samostatný proces a je zcela nezávislý na modulech ostatních. To umožňuje kdykoliv přidat nebo odebrat některý z těchto modulů a případně knihovnu rozšířit o moduly nové.

Všechny vytvořené moduly implementují přetíženou metodu *ToString*, která následně umožňuje snazší práci s výstupem aplikace.

Některé často používané funkcionality pro práci s řetězci a kolekcemi jsou implementovány formou externích rozšíření.

Moduly Implementovány byly následující moduly. Pořadí popisu modulů pak odpovídá jednotlivým krokům statické analýzy.

MIME Prvním krokem statické analýzy je detekce typu MIME. Tu zajišťuje knihovna *MIME* viz kapitola 8.1.

Nejprve je vytvořena instance třídy pro zjištění tohoto typu a poté se zavolá potřebná metoda, jež slouží k zjištění typu.

Hashes Tento modul zajišťuje výpočet několika různých hašovacích funkcí, a to konkrétně MD5, SHA-1, SHA-256, SHA-384, SHA-512. Tyto funkce slouží primárně pro možnost případného vyhledání vzorku v různých databázích malwaru.

Z veřejných databází lze využít například databázi *Malshare.com*, případně již dříve zmíněný VirusTotal.

VirusTotal Dalším modulem je modul pro aplikaci VirusTotal, předání pak zajišťuje knihovna *VirusTotalNET* (viz kapitola 8.1).

Zpracování a odeslání dat službě VirusTotal, je velmi jednoduché a to proto, že většinu funkční části zajišťuje samotná knihovna.

Nejprve je vytvořena instance třídy *VirusTotal*. Pro komunikaci se službou VirusTotal je zapotřebí mít vytvořený účet v této službě a zaregistrovat si vlastní API klíč (jedná se o klíč pro programovatelné rozhraní služby, jež slouží k autorizaci požadavků). V základním režimu je možné odeslat k testování 4 požadavky za minutu a 1000 požadavků denně.

Pokud již byla vytvořena instance, může následovat testování daného vzorku. Nejprve je provedeno vyhledání v databázi již existujících výsledků testů. Pokud vzorek testován dosud nebyl, je odeslán k testu. Dalším omezením může být velikost souboru. Její horní hranice ve veřejném API je 32 MB.

Entropie Následuje krok, ve kterém je vypočtena entropie pro zkoumaný soubor. Výsledek pak může naznačovat, zda bylo provedeno šifrování nebo komprese souboru [2].

V daném případě se provede výpočet pomocí následujícího vzorce 2, který je aplikován na načtený soubor v paměti jako pole bytů.

$$En = -\sum_{n=1}^{\infty} f_n * \frac{\log f_n}{\log 2} \tag{2}$$

$$f_n = \frac{p}{c} \tag{3}$$

Kde $p =$ počet výskytu stejného bytu a $c =$ celková délka pole bytů.

Čím více se výsledná hodnota entropie (En) blíží číslu 8, tím více je pravděpodobné, že se jedná o šifrovaný nebo komprimovaný soubor [75].

Ukázku výsledné entropie prezentuje následující tabulka č. 5. Pro testování byl zvolen multimediální přehrávač VLC. Nejdříve bylo provedeno packování pomocí známého packeru pro kompresi UPX a komerečního packeru ASPack a následně byla vypočtena entropie jednotlivých souborů.

	Entropie	Velikost souboru
bez komprese	6.47	940 kB
UPX	7.83	324 kB
ASPack	7.85	316 kB

Tabulka 5: Ukázka entropie packerů UPX a ASPack

PE Modul PE zajišťuje analýzu hlavičky spustitelných souborů PE (viz kapitola č. 4.2) a to za pomoci knihovny PeNET (viz kapitola 8.1).

Nejprve je generována instance třídy *PeFile* z knihovny PeNET. Ta zajišťuje veškerou práci se strukturou PE souboru, jako jsou zjištění jednotlivých sekcí, importů knihoven, funkcí atd.

Následně se pro zjednodušení práce s PE strukturou načtou do vlastní třídy modulu *PE* importy, exporty, directories a sekce (viz kapitola 4.2).

Modul obsahuje implementaci pro zjištění jak základních parametrů jako je adresa vstupního bodu (angl. entry point) programu, velikost, čas vytvoření, typ a zda se jedná o knihovnu DLL, ovladač nebo EXE soubor. Ale také pokročilejší informace o importovaných a exportovaných funkcích a knihovnách, seznam sekcí a jejich parametrů nebo seznam jednotlivých directories.

Ukázku výstupu lze vidět na následujícím obrázku 12.

Obrázek 12: Ukázka výstupu informací o spustitelném souboru formátu PE

```
# pe "E:\Steam\steamapps\common\Grand Theft Auto V\GTA5.exe"
PE report:
file size: 77330048
image base: 5368709120
entry point: 24657944
import hash: 085b60b18abd6eb18a5588c0a665b55d
time date stamp: 1585257826
signed: True
signsuer: CN=DigiCert SHA2 Assured ID Code Signing CA, OU=www.digicert.com, O=DigiCert Inc, C=US
sign subject: CN="Rockstar Games, Inc.", OU=Rockstar Games, O="Rockstar Games, Inc.", L=New York, S=New York, C=US
date time: 26.03.2020 22:23:46
machine: Amd64
filename: E:\Steam\steamapps\common\Grand Theft Auto V\GTA5.exe
dll: False
exe: True
driver: False
32b: False
64b: True
dot net: False
sections: .text, BINK, BINKBSS, .rdata, .data, .pdata, .tls, BINKCONS, .rsrc, .reloc, .text
imported dlls: KERNEL32.dll, USER32.dll, steam_api64.dll, GFSDK_ShadowLib.win64.dll, WS2_32.dll, VERSION.dll, DSOUND.
 PSAPI.DLL, MF.dll, MFPlat.DLL, msdmo.dll, MFReadWrite.dll, PROPSYS.dll, CRYPT32.dll, WINTRUST.dll, WTSAPI32.dll, d3d
SDK_TXAA_AlphaResolve.win64.dll, IMM32.dll, DINPUT8.dll, XINPUT1_3.dll, RPCRT4.dll, IPHLPAPI.DLL, SHLWAPI.dll, GDI32.
l, ole32.dll, OLEAUT32.dll
exports:
directories: Import, Resource, Exception, Security, BaseReloc, Debug, TLS, IAT
```

Strings Analýza řetězců probíhá v modulu *Strings*. Průběh analýzy je rozdělen do několika kroků. Nejprve je načten soubor do paměti a převeden na řetězec. Poté je provedena filtrace nepožadovaných znaků (netisknutelné znaky, konec řetězce atp.), tyto znaky jsou odstraněny z toho důvodu, aby byla zajištěna lepší čitelnost výstupu při ruční analýze.

Následně je vyhledán název souborů uložených v programu pomocí vygenerovaného regulárního výrazu. Ten je vytvořen na základě získaného seznamu MIME typů (přípon souboru). Tento seznam je nejdříve načten do paměti z JSON souboru. Poté je provedeno vygenerování regulárního výrazu *..(.jpg|.png ... |.exe)..* a vytvořena instance pro vyhledání jednotlivých výskytů v testovaném souboru. Zjištěná data jsou vrácena formou seznamu.

Dále je provedeno vyhledání důležitých informací, které mohou sloužit malwaru jako způsob, jak informovat útočníka o stavu útoku nebo naopak jako zdroj úkolu pro program, například vzdálená aktivace DDoS (Denial of service - útok s cílem zahltit požadovaný cíl) apod. Konkrétně se může jednat o IP adresy, emaily nebo informace ve formátu URL adresy jako *https://*, *smb://* nebo například *ftp://*.

Vyhledání těchto dat se opět provádí pomocí regulárního výrazu, v tomto případě se pak jedná o předem vytvořené a uložené konstanty.

Posledním krokem analýzy je vyhledání známých metod používaných malwarem. Jako je například volání Windows API, Anti-debuggování atd. Seznam těchto metod byl získán z GitHub repositáře *Cisco-Talos/Bass*. Vyhledání těchto metod je prováděno pomocí metody *Contains*. Pokud je požadovaný řetězec obsažen, přidá se do seznamu.

Detect with Yara Pro detekci dalších vlastností byl vytvořen modul, který zajišťuje analýzu pokročilejších vlastností jako například kontrolu signatur různých packerů, výskyt různých metod chování na základě obsažené kombinace knihoven nebo detekci obranných mechanismů vůči analýze.

Pro tuto detekci, jsou použity již vytvořená pravidla z veřejného repositáře *Yara-rules/rules* na platformě pro sdílení GIT repositářů Github.

A jsou to konkrétně pravidla pro detekci důležitých vlastností jako například obrana vůči debuggování nebo virtuálnímu prostředí. Dále to pak jsou pravidla pro detekci známých signatur packerů nebo kompilátorů.

8.3 Výstup aplikace

Pro zajištění výstupu je v knihovně *StaticAnalysisProject.Lib* implementována třída *FileReport*. Třída *FileReport* zajišťuje veškerou inicializaci instancí jednotlivých modulů a zpracování dat z jednotlivých části statické analýzy testovaného vzorku.

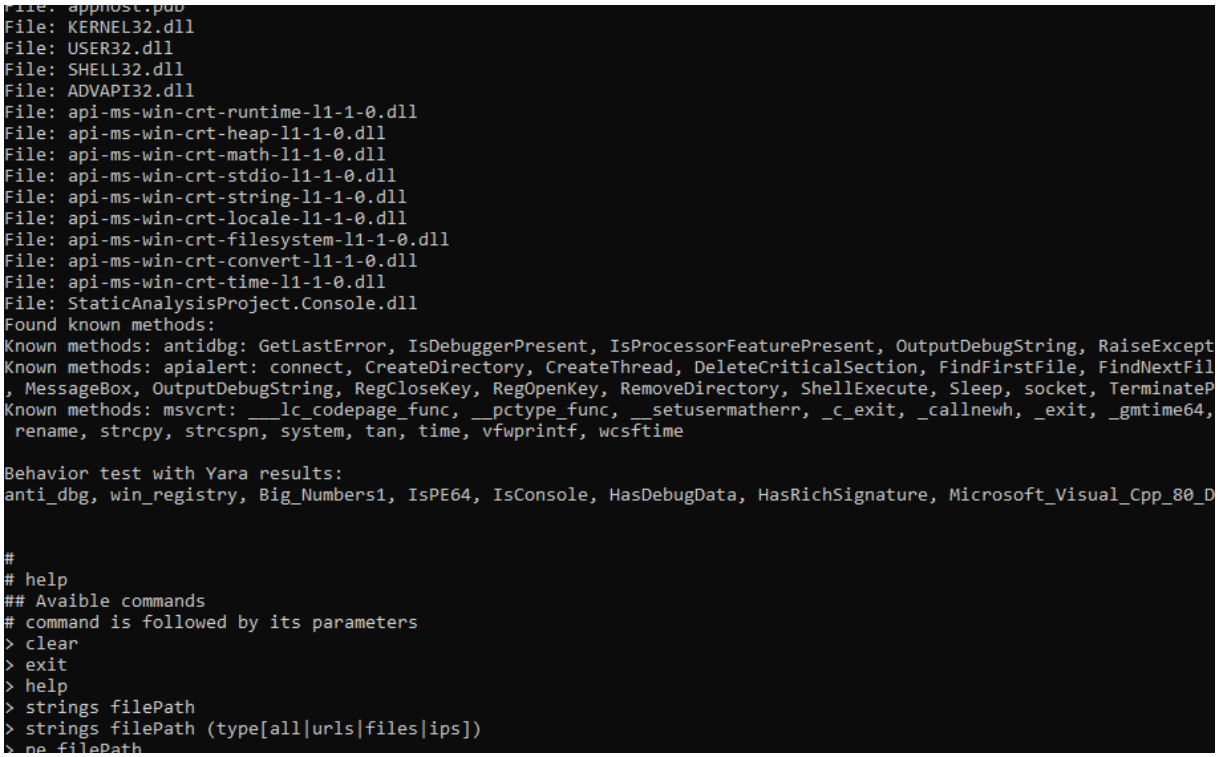
Řešení pak obsahuje implementaci dvou odlišných přístupů pro výstup aplikace. A to jako webová nebo konzolová aplikace. Ty jsou popsány v následujícím textu.

Konzolová aplikace Rozhraní konzolové aplikace je koncipováno jako interaktivní uživatelský příkazový řádek. Po spuštění aplikace je uživatel uveden do prostředí programu, kde pomocí zápisu textových příkazů může tuto aplikaci ovládat.

Příkazy v programu jsou uzpůsobeny tak, že je možné provést buď jednotlivé kroky statické analýzy pomocí modulů této aplikace (viz kapitola 8.2), anebo provést kompletní analýzu a získat tak výstup.

Na následujícím obrázku č. 13 lze vidět výstup interaktivního prostředí konzolové aplikace.

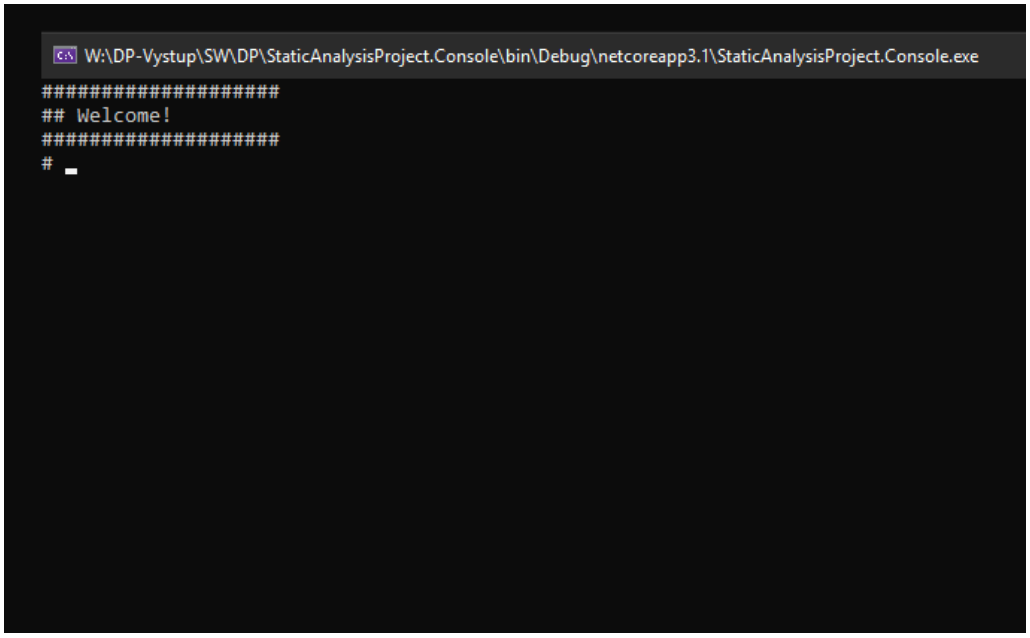
Obrázek 13: Ukázka výstupu konzolové aplikace



Další možností je pak spuštění aplikace s parametry, kdy lze získat přímý výstup bez nutnosti vstupovat do interaktivního prostředí programu.

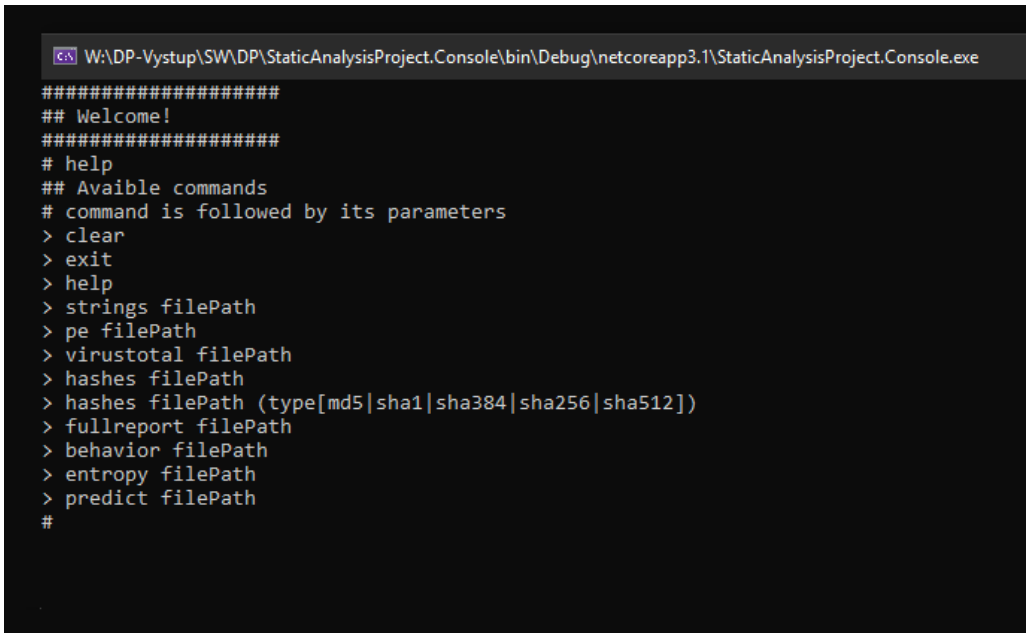
Práce s aplikací Po spuštění se uživatel ocitne v interaktivním prostředí konzolové aplikace, kde může začít zadávat příkazy viz obrázek č. 14.

Obrázek 14: Práce s konzolovou aplikací - spuštění



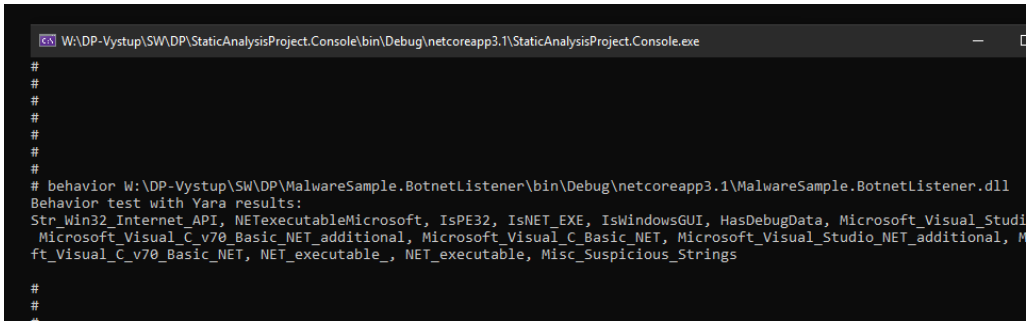
Prvním příkazem, který může uživatel využít je příkaz *help*, který zobrazí nápovědu (seznam použitelných příkazů) (viz obrázek č. 15).

Obrázek 15: Práce s konzolovou aplikací - příkaz help



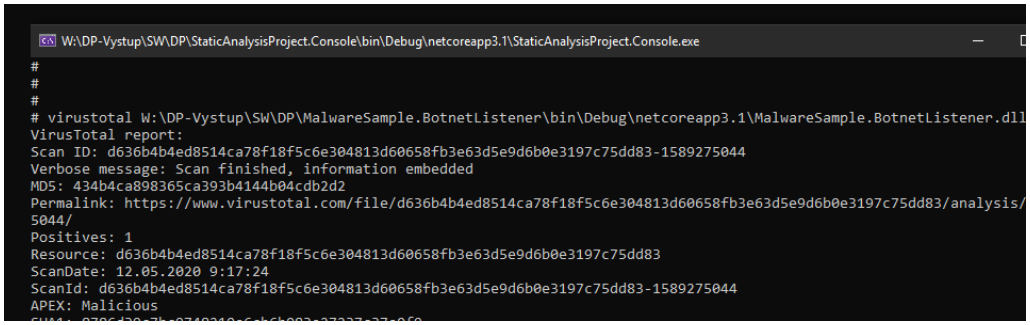
Pokud je potřeba provést analýzu pomocí nástroje *Yara*, použijeme příkaz *behavior* viz obrázek č. 16. Výstupem pak je seznam pozitivních pravidel, které mohou naznačovat chování aplikace.

Obrázek 16: Práce s konzolovou aplikací - test pomocí Yara



Dalším příkazem pak může být testování službou *VirusTotal*, která provede skenování různými antivirovými programy. Výstup lze vidět na obrázku č. 17

Obrázek 17: Práce s konzolovou aplikací - odeslání na VirusTotal



Webová aplikace Hlavním cílem vývoje webové aplikace, je možnost jednoduše bez nutnosti instalace dalšího softwaru provést statickou analýzu.

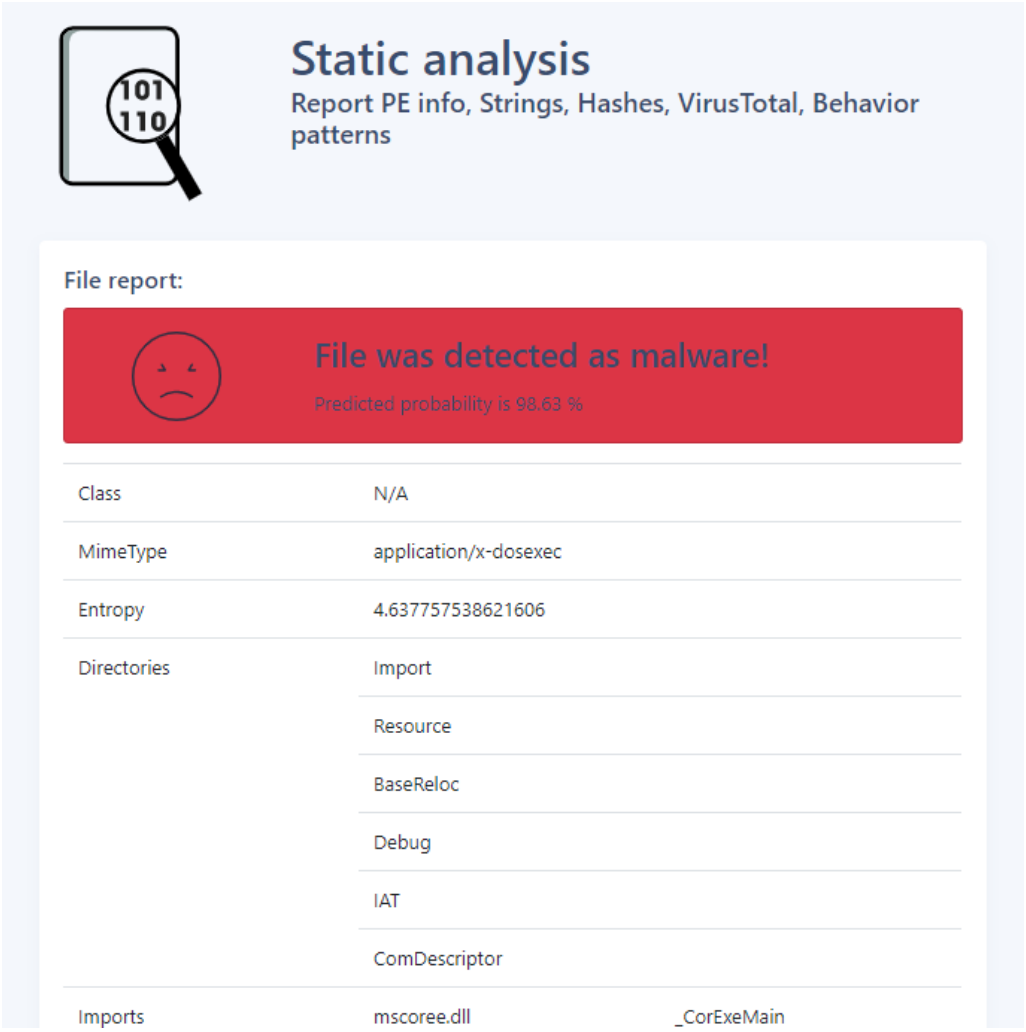
Uživatelské rozhraní je velmi jednoduché, po otevření webové stránky v prohlížeči stačí zvolit požadovaný soubor. Následně je provedeno odeslání na webový server, kde je provedena statická analýza obdobně jako u konzolové aplikace s plným výstupem.

Uživatel tak může vidět kompletní výstup statické analýzy. Součástí výstupu je také výsledek binární klasifikace provedené pomocí knihovny ML.NET (viz kapitola 8.1). Průběh klasifikace je popsán dále v kapitole 8.4.

Při vývoji webové aplikace byly použity moderní technologie jako je framework jQuery, který umožňuje snazší vývoj interaktivních webových aplikací. Dále pak rozšíření standardních kaskádových stylů o dynamické funkce LESS a CSS framework Bootstrap ve verzi 4.

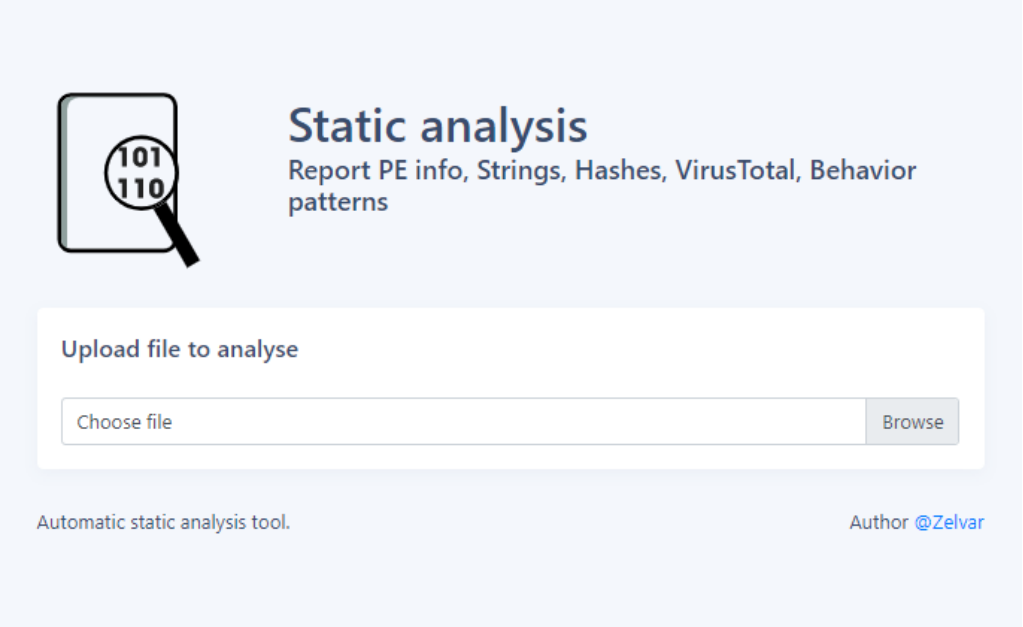
Na následujícím obrázku č. 18 je možné vidět ukázku výstupu webové aplikace. V první části je zvýrazněn výstup klasifikace, který byl dle predikované pravděpodobnosti barevně rozlišen (červená pokud je detekován malware; jestliže malware detekován nebyl a zároveň je pravděpodobnost vyšší než 80 % je výsledek zobrazen zeleně; v případě, že je pravděpodobnost nižší než 80 % je zvýrazněn žlutě; pokud má výsledek nižší pravděpodobnost než 20 % je výsledek šedý).

Obrázek 18: Ukázka výstupu webové aplikace



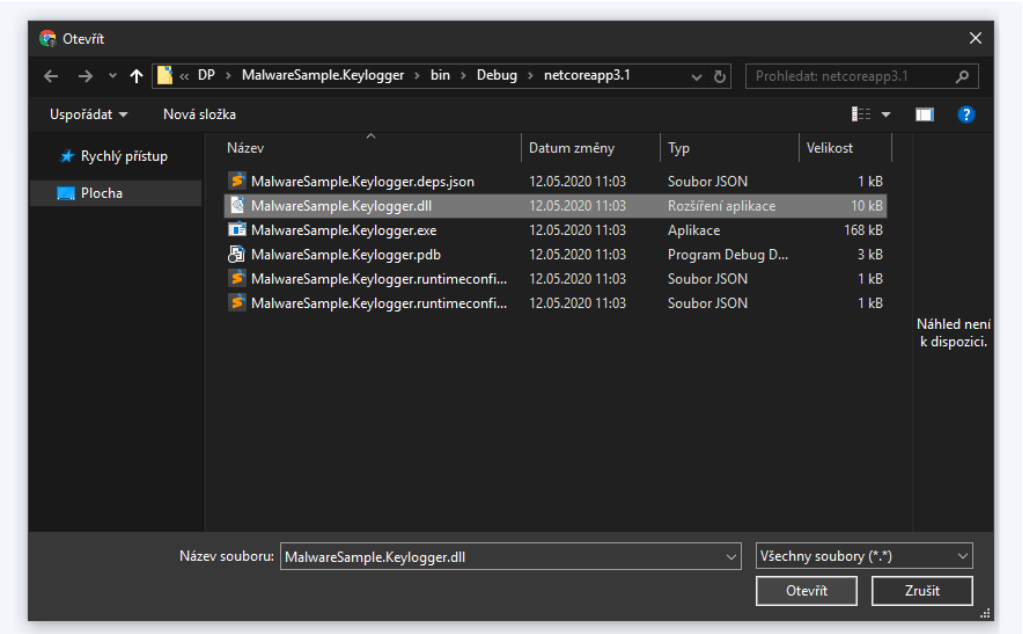
Práce s aplikací Po vstupu na webovou stránku je uživateli zobrazen webový formulář, kde je vyzván k výběru požadovaného souboru viz 19.

Obrázek 19: Práce s webovou aplikací krok 1.



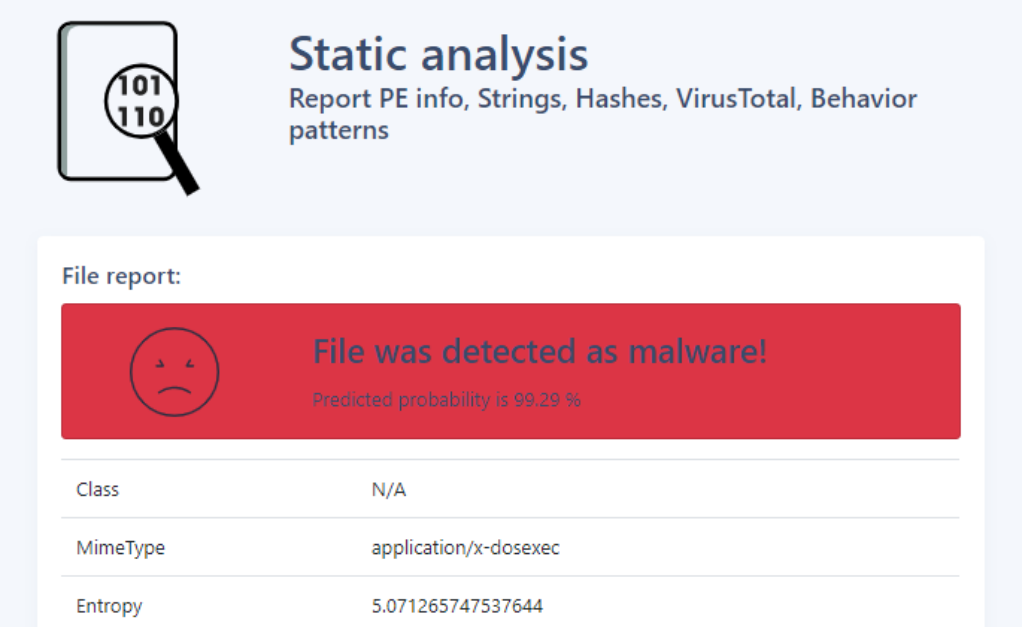
Následně je vybrán požadovaný soubor pro testování viz obrázek č. 20 a poté potvrzen výběr souboru.

Obrázek 20: Práce s webovou aplikací krok 2.



Aplikace poté zpracuje analýzu požadovaného souboru a zobrazí ji uživateli, což je vidět na obrázku č. 21.

Obrázek 21: Práce s webovou aplikací krok 3.



8.4 Testování aplikace

V průběhu vývoje programu byla aplikace testovaná na různých kategorií souborů. Především bylo testování zaměřeno na malware. Testování je pak konkrétněji popsáno dále v této kapitole.

Testování malwaru Testování malwaru bylo prováděno formou analýzy zvolených souborů. Průběh této analýzy je zmíněn již v kapitole 8.2. Tato kapitola se dále zaměří pouze na testování malwaru určeného pro operační systém Windows ve formátu PE (viz kapitola 4.2).

Aby analýza malwaru mohla být vůbec provedena, je nejprve zapotřebí vytvořit nebo získat vzorky malwaru.

Za tímto účelem je z veřejně dostupných databází malwaru získáno 1952 vzorků, které jsou následně automaticky analyzovány pomocí vytvořeného programu. Tento malware také dále slouží jako vhodný zdroj informací pro vytvoření kolekce dat (angl. dataset), který má i funkci vstupního zdroje informací pro strojové učení.

Již zmíněné databáze malwaru lze nalézt na webové adrese `https://dasmalwerk.eu` a dále pak na GitHubu, konkrétně v repositáři *ytisf/theZoo*.

Tak, aby bylo možné maximálně automatizovat skenování velkého množství dat, je vytvořen jednoduchý program. Tento program zajišťuje vyhledání souborů v požadované cestě určené při spuštění programu.

Program zároveň zajišťuje aby nedošlo k překročení omezení API služby *VirusTotal*, která v základním tarifu nedovoluje více než 4 požadavky za minutu a 1000 požadavků za jeden den. V základním režimu je také omezena velikost souboru na 32 MB [76].

Nakonec se výsledky analýzy zaznamenávají do jednoho souboru ve formátu *JSON*, který je následně použit jako zdroj informací.

Pro porovnání bylo do datasetu také vloženo 108 výstupu legitimních aplikací (dále nazývané jako software). Při výběru je brán zřetel na nutnost různorodosti jednotlivých vlastností. Proto je vybráno několik podskupin jako například hry, správce souborů, kancelářské programy ale také peněženky pro virtuální měny atd. Tato část kolekce je dále nazývána jako software.

Následující tabulka č. 6 prezentuje podíl spustitelných souborů pro OS Windows, detekovaný pomocí modulu *MIME* (viz kapitola 8.2) obsažený v datové kolekci.

Tabulka 6: Podíl spustitelných souborů v datasetu

	Software	Malware
PE (.exe, .dll)	108	769
jiný	0	1183
celkem	108	1952

Vzhledem k tomu, že se práce zabývá primárně malwarem určeným pro operační systém Windows, jsou dále zohledňována a uváděna data pouze pro malware útočící právě na tento OS.

Tabulka č. 7 demonstruje podíl jednotlivých parametrů spustitelných PE souborů v datové kolekci. Parametry *16b*, *32b* a *64b* určují pro jakou architekturu procesoru je program určen, zda se jedná o 32 nebo 64 bitový systém. Vzhledem k tomu, že 64 bitový systém podporuje také spouštění 32 bitových aplikací. Útočnici mohou stále vytvářet škodlivý kód pro zajištění kompatibility s 32 bitovou verzí systému. Zároveň lze však dle statistik *PassMark Software* očekávat sestupný podíl malwaru pro 32 bitovou platformu. A to převážně z důvodu klesajícího počtu instalací 32 bitových Windows [77].

Podíl podepsaného (Signed) škodlivého kódu nebo malwaru skrytého jako ovladače není často velký. Část nativního kódu převažuje nad kódem interpretovaným alespoň v případě platformy .NET.

Tabulka 7: Parametry spustitelných souborů v kolekci dat

	Software	Malware
16b	0.00 %	1.56 %
32b	50.00 %	96.10 %
64b	50.00 %	2.34 %
DLL	21.30 %	5.85 %
EXE	78.70 %	91.81 %
.NET	4.63 %	15.08 %
Driver	0.00 %	0.13 %
Signed	33.33 %	11.44 %

Další tabulka č. 8 pak prezentuje střední hodnoty počtů sekcí spustitelných PE souborů v jednotlivých skupinách. Na základě těchto dat lze usuzovat, že malware disponuje průměrně menším počtem sekcí než běžný software.

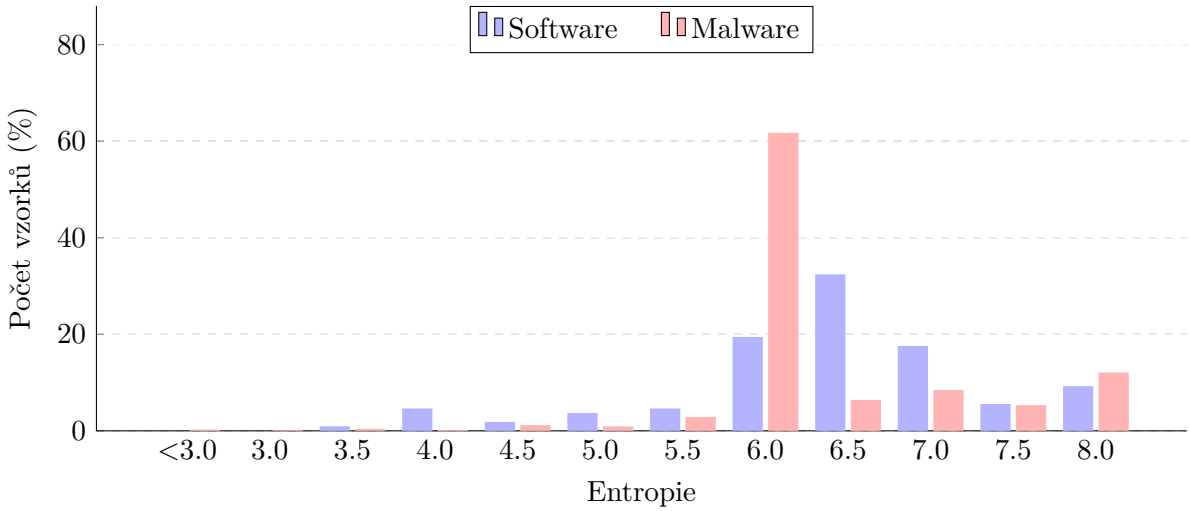
Tabulka 8: Počty sekcí v jednotlivých skupinách

	Software	Malware
Průměr	7.34	4.81
Medián	7.00	4.00
S.D.	3.03	1.9

Tabulka 9: Entropie jednotlivých skupin

	Software	Malware
Průměr	6.38	6.97
Medián	6.41	7.08
Směrodatná odchylka	0.94	1.00

Následující graf na obrázku č. 22 obsahuje četnosti hodnot entropie pro software a malware. Pro lepší orientaci byla entropie zaokrouhlena v kroku 0.5. Hodnoty entropie softwaru vykazují na hladině spolehlivosti 98 % normální rozložení naopak u hodnot entropie pro malware lze pozorovat extrém entropie v hodnotě 6.0 a druhý v hodnotně 8.0. Zároveň je z tabulky 9 patrné, že průměrné hodnoty entropie a také jejího mediánu jsou pro malware vyšší než pro software. To, že se hodnoty statisticky významně liší, bylo prokázáno i za použití Mann-Whitneyova U-testu.



Obrázek 22: Entropie vzorků

Dalším zjišťovaným parametrem (viz tabulka č. 10) byla informace, jež určovala zda testovaný vzorek obsahuje konkrétní typ řetězce. V tomto případě se jednalo o *email*, *ip adresu* (jak IPv4 tak IPv6) a *url adresu*. Zajímavým zjištěním je, že v případě malwaru je výskyt *emailové* a *url* adresy nižší. V případě malwaru mohou tyto informace, které slouží často ke komunikaci s útočníkem skryté pomocí různých obfuskačních metod viz kapitola 6.2.

Tabulka 10: Výskyt konkrétního typu řetězce

	Software	Malware
Mail	53.70 %	26.80 %
IP	94.40 %	87.30 %
URL	85.20 %	34.30 %

Následující data v tabulce č. 11 prezentují patnáct nejpoužívanějších knihoven v kolekci dat, používané jak softwarem tak malwarem. Na datech můžeme jasně vidět, že se použité knihovny se velmi podobají a rozlišit malware od legitimních aplikací (software) pouze pomocí této informace není možné. Proto se tato informace kombinuje se seznamem použitých metod, které jsou volány z těchto knihoven.

Tabulka 11: Porovnání výskytu použitých knihoven

Software		Malware	
Knihovna	Výskyt	Knihovna	Výskyt
kernel32.dll	92.59 %	kernel32.dll	76.98 %
user32.dll	75.00 %	user32.dll	60.99 %
advapi32.dll	73.15 %	advapi32.dll	59.82 %
shell32.dll	68.52 %	gdi32.dll	38.75 %
gdi32.dll	57.41 %	shell32.dll	36.80 %
ole32.dll	56.48 %	ole32.dll	35.37 %
oleaut32.dll	44.44 %	comctl32.dll	33.29 %
comctl32.dll	38.89 %	oleaut32.dll	23.41 %
version.dll	37.96 %	version.dll	21.07 %
shlwapi.dll	32.41 %	ws2_32.dll	19.25 %
winmm.dll	24.07 %	mscoree.dll	14.95 %
ws2_32.dll	22.22 %	msvcrt.dll	14.95 %
comdlg32.dll	21.30 %	shlwapi.dll	13.26 %
vcruntime140.dll	18.52 %	wininet.dll	13.26 %
api-ms-win-crt-runtime-l1-1-0.dll	18.52 %	comdlg32.dll	9.36 %

Seznam patnácti nejpoužívanějších metod je pak možné vidět v následující tabulce č. 12,

kde je opět vidět jistou podobnost mezi softwarem a malwarem. Rozdíl ve volaných funkcích je však větší. Prvním příkladem může být metoda *GetModuleHandleA*, kterou malware využívá k nalezení vhodného místa pro injekci vlastního kódu [78]. Příkladem může být například injekce vlastního kódu do kernelu (*GetModuleHandleA(kernel32.dll)*). Dalším metodou častěji používanou malwarem je *LoadLibraryA*. Tato metoda slouží k dynamickému načtení modulu do adresního prostoru aplikace. Často je také spolu s funkcí *CreateRemoteThread* využívána při technice zvané *DLL Injection* [79]. Tato technika spočívá ve vložení (tzv. injekci) vlastního kódu do běžícího procesu. Tímto postupem může útočník injektovat malware například do procesů operačního systému.

Tabulka 12: Porovnání výskytu použitých metod pro malware a software

Software		Malware	
Funkce	Výskyt	Funkce	Výskyt
GetProcAddress	96.30 %	GetProcAddress	81.79 %
CloseHandle	91.67 %	WriteFile	67.75 %
GetCurrentThreadId	91.67 %	GetModuleHandleA	65.54 %
GetLastError	91.67 %	Sleep	64.63 %
Sleep	90.74 %	CloseHandle	64.50 %
GetModuleHandleW	90.74 %	GetLastError	63.20 %
QueryPerformanc..	89.81 %	ExitProcess	62.03 %
GetCurrentProcess	89.81 %	LoadLibraryA	59.17 %
WriteFile	87.96 %	MultiByteToWideChar	56.31 %
WideCharToMultiByte	87.04 %	GetModuleFileNameA	52.15 %
MultiByteToWideChar	87.04 %	GetTickCount	52.15 %
FreeLibrary	87.04 %	WideCharToMultiByte	49.80 %
UnhandledExcept..	83.33 %	FreeLibrary	49.02 %
GetCurrentProcessId	81.48 %	GetCurrentProcess	47.20 %
GetModuleFileNameW	80.56 %	RegCloseKey	46.68 %
RegCloseKey	78.70 %	VirtualAlloc	46.03 %
TerminateProcess	77.78 %	WaitForSingleObject	43.17 %
EnterCriticalSection	77.78 %	GetStdHandle	43.04 %
LeaveCriticalSection	77.78 %	ReadFile	42.91 %
GetSystemTimeAs..	76.85 %	SetFilePointer	42.00 %

Výskyt detekovaných pravidel prezentuje tabulka č. 13. Na výsledných datech lze vidět procentuální výskyt patnácti nejčastějších pravidel, které byly detekovány v obou skupinách. V případě malwaru můžeme jasně vidět, že bylo nejčastěji detekováno pravidlo (*IsPE32*), které detekuje 32 bitový spustitelný soubor ve formátu PE. Následují pravidla pro detekci výskytu grafic-

kého rozhraní pro Windows *IsWindowsGUI* a pravidlo pro detekci manipulace s práce se soubory pomocí Windows API (*win_files_operation*). Oproti běžnému softwaru, bylo u malwaru častěji detekováno pravidlo *IsPacked*, které naznačuje, že u malwaru bylo detekován zabalení packerem (viz kapitola 6.4). Dále pak bylo detekováno pravidlo *SEH_Init* a *SEH_Save*, které značí, že zdrojový kód aplikace obsahuje inicializaci pro strukturovanou obsluhu výjimek. V tomto případě se může jednat o jeden ze způsobu ochrany proti ladění (anti-debug viz kapitola 6.1). Dalším důležitým pravidlem, které bylo detekováno je *escalate_priv*. Toto pravidlo detekuje způsoby jakým se útočník snaží získat vyšší oprávnění a to tak, aby bylo možné vykonat jinak nerealizovatelný útok [80].

Tabulka 13: Porovnání detekovaných YARA pravidel

Software		Malware	
Pravidlo	Výskyt	Pravidlo	Výskyt
IsWindowsGUI	75.00 %	IsPE32	95.97 %
win_files_operation	64.81 %	IsWindowsGUI	93.37 %
HasOverlay	60.19 %	win_files_operation	58.00 %
win_registry	59.26 %	HasRichSignature	56.18 %
anti_dbg	57.41 %	IsPacked	52.93 %
HasRichSignature	50.00 %	win_registry	44.21 %
HasDebugData	50.00 %	SEH_Init	43.82 %
IsPE32	45.37 %	HasOverlay	36.02 %
screenshot	42.59 %	screenshot	33.29 %
CRC32_poly_Constant	42.59 %	CRC32_poly_Constant	30.43 %
IsPE64	39.81 %	SEH_Save	29.78 %
win_token	37.04 %	MS_VS_Cpp_v50..	28.09 %
win_mutex	37.04 %	Str_Win32_Wins..	27.05 %
HasDigitalSignature	34.26 %	win_token	26.14 %
keylogger	31.48 %	escalate_priv	22.11 %

Vlastní malware Pro účely testování jsou jako součást práce vytvořeny dva jednoduché druhy malwaru se zcela rozdílným vektorem útoku. Jednotlivé vlastnosti těchto škodlivých programů jsou popsány níže. Tyto vzorky byly primárně použity pro testování klasifikace. Výsledky těchto testů budou popsány dále.

Oba tyto testovací vzorky jsou implementovány v technologií .NET Core jako aplikace s grafickým rozhraním (zkr. GUI) a to i přes to, že se při spuštění škodlivého programu žádné GUI nevytváří.

Botnet První ze dvou vzorků funguje na principu velmi jednoduchého klienta sítě, který je vzdáleně ovládán pomocí *CnC* serveru. Klient může vykonávat pouze jednoduchý druh vyřadit službu z provozu (tzv. Denial of service), a to pomocí zasílání pingu z příkazového řádku. V případě vytvoření větší sítě klientů by se pak mohlo jednat o distribuovaný druh útoku. Princip byl velmi zjednodušen s cílem vytvořit funkční imitaci malwaru.

Celá síť se ovládá pomocí *Command-and-Control* serveru, jež byl implementován jako *XML* (angl. Extensible Markup Language) dokument umístěný na webovém serveru. Tento dokument obsahuje cíl útoku (IP adresu) a informaci o aktivaci (zapnuto nebo vypnuto).

Keylogger Druhým škodlivým programem je pak aplikace pro skryté zaznamenávání stisku klávesnice a odesílání na webový server. Aplikace se po spuštění zavěsí na potřebné *Windows API*, které slouží k získávání informací o stisku kláves a následně je vyvolávána při stisku volitelné klávesy událost, jež má za úkol zaznamenat stisknutý znak do proměnné.

Pokud již bylo zaznamenáno 100 znaků, informace (zaznamenané stisky kláves) se odešlou na webový server, kde jej jednoduchý *PHP* skript zaznamená do textového souboru.

V případě, že uživatel například použije platební kartu na internetu a útočník zaznamená jeho stisky kláves, mohou být jím zadané údaje následně zneužity.

Klasifikace výstupu analýzy Klasifikace výstupu probíhala pomocí výše zmíněné technologie *ML.NET* (viz kapitola 8.1), která umožňuje snadné nasazení strojového učení na téměř jakoukoliv aplikaci.

Pro nasazení strojového učení stačí přidat do modelu, který obsahuje data, k jednotlivým proměnným parametr *[LoadColumn(x)]* (kde x je pořadí sloupce). Tím zajistíme možnost využít data pro strojové učení. Protože však technologie podporuje pouze některé datové typy (řetězce, boolean, float, vektorové data atd.) bylo potřeba data předzpracovat do požadovaného formátu. Proto byl model značně zjednodušen a obsahuje pouze některé parametry z výstupu statické analýzy.

Těmito parametry jsou základní vlastnosti PE hlavičky jako typ (Ovladač, *DLL*, *EXE*, platforma), zda je aplikace podepsaná digitálním certifikátem, třída (malware nebo software), entropie, detekované řetězce (*email*, *IP* a *URL*), počet pozitivních testů *VirusTotal*, seznam používaných metod a knihoven a zjištěné pozitivní výsledky pravidel *Yara*.

Následně je provedeno automatické trénování (učení s učitelem) klasifikačního modelu pomocí vytvořené kolekce dat. Pro trénování je použito 80 % dat datové kolekce, zbylých 30 % je použito pro křížovou validaci s deseti opakováními. Křížová validace má za úkol zjistit (vyhodnotit) jak moc jsou ovlivňovány nezávislé vzorky dat [81].

Pro trénování byl použit algoritmus *SdcaLogisticRegression*, který má zajistit dobré výsledky bez nutnosti jej dále ladit [82]. Aby bylo možné tento algoritmus použít, je potřeba data nejdříve normalizovat. Pro tuto transformaci se využívá metoda *NormalizeBinning*.

Součástí vypracovaní této diplomové práce je již vygenerovaný model na základě datové kolekce, která je také součástí. V případě však, že dojde ke změně dat je nutné tento model odstranit a nechat jej program vygenerovat znovu.

Následně se tento vytrénovaný model použije pro predikci třídy (software nebo malware), testovaného vzorku na základě výstupu statické analýzy.

Testování Pro testování byly použity výše zmíněné vzorky malwaru, které byly vytvořeny jako součást této práce (viz kapitola 8.4). A dvacet legitimních aplikací a dvacet dalších vzorků malwaru z databáze *VirusShare.com*. Tyto vzorky nebyly použity při vytváření datové kolekce.

Následující tabulka č. 14 prezentuje výsledky testování vlastního malwaru. Jak lze vidět, malware byl detekován s přesností přes 90 %. A oba vzorky byly pozitivně testovány. Úspěšnost je tedy 100 %.

Soubor	Detekováno	Pravděpodobnost
KeyLogger.dll	Ano	91.34 %
BotnetListener.dll	Ano	93.71 %

Tabulka 14: Detekce vytvořeného malwaru

Další testování je provedeno na malwaru z databáze *VirusShare.com*. Z této databáze je staženo 20 vzorků, které jsou podrobeny testům. Tabulka č. 15 obsahuje část kontrolního součtu vytvořeného pomocí *SHA-256* (kompletní kontrolní součty jsou pak součástí přílohy), název pod kterým byl detekován antivirem *Avast*, zda byl detekován a s jakou pravděpodobnosti. V tomto případě je úspěšnost testování 100 %.

SHA256	Avast	Detekováno	Pravděpodobnost
cbfc9d3..	Undetected	Ano	99.99 %
8757088..	Undetected	Ano	99.99 %
000ad60..	Win32:PUP-gen [PUP]	Ano	99.99 %
037bfec..	Win32:AdwareSig [Adw]	Ano	100 %
0872a6c..	Win32:VBCrypt-AKV [PUP]	Ano	99.99 %
190ea43..	Win32:Downloader-GYZ [Trj]	Ano	99.99 %
1c757b4..	Win32:Adware-CWL [PUP]	Ano	99.32 %
21b0957..	Win32:Adware-gen [Adw]	Ano	99.99 %
2891ae4..	Win32:Adware-gen [Adw]	Ano	99.91 %
2e58e15..	Win32:AdwareSig [Adw]	Ano	99.99 %
436b7f2..	Win32:Vundo-GZ [Trj]	Ano	99.77 %
56bb2f5..	Win32:AdwareSig [Adw]	Ano	00.00 %
78c38d9..	Win32:Dropper-OYB [PUP]	Ano	99.99 %
97f1adc..	Win32:AdwareSig [Adw]	Ano	00.00 %
a2cb313..	Win32:RmnDrp	Ano	99.81 %
abaa67c..	Win32:AdwareSig [Adw]	Ano	99.99 %
b154f80..	Win32:AdwareSig [Adw]	Ano	99.99 %
bf5b884..	Win32:Mirc-AA [PUP]	Ano	99.99 %
c494dab..	Win32:Malware-gen	Ano	99.98 %
db1ca5a..	Win32:Malware-gen	Ano	9.98 %

Tabulka 15: Detekce malwaru z databáze VirusShare.com

Poslední testování je provedeno nad sadou nepoužitých legitimních aplikací v datové kolekci. Výsledky tohoto testování prezentuje tabulka č. 16. V případě 3 vzorků (15 % testů) dochází k predikcí tzv. *false-positive* výsledku. Tento jev může být způsoben přecitlivělým predikčním modelem a bylo by vhodné datovou kolekci dále rozšířit.

SHA256	Název souboru	Detekováno	Pravděpodobnost
85601e9..	Ozone Neon M50 Driver.exe	Ano	99.05 %
9299fb2..	Curse.exe	Ano	64.86 %
baf1879..	Lanayo.VagrantManager.exe	Ano	99.04 %
775a767..	Amazon.Fuel.AppCore.dll	Ne	4.46 %
dcdb065..	CAMV2.Data.dll	Ne	5.12 %
459dd1e..	EasyAntiCheat.exe	Ne	3.39 %
47772a9..	NeroPatentActivation.exe	Ne	4.79 %
f2d6f5b..	lua52.dll	Ne	0.00 %
c1c6c4c..	mergecap.exe	Ne	0.00 %
280c0f7..	obs32.exe	Ne	0.00 %
10effd4..	OpenRL.dll	Ne	0.00 %
3ad5d5b..	overlay.dll	Ne	0.00 %
b3e7e2e..	phpstorm.exe	Ne	40.39 %
4838635..	python.exe	Ne	0.05 %
f4aae23..	Qt5Core.dll	Ne	0.00 %
a64fe80..	socialclub.dll	Ne	0.00 %
41229db..	SocialClubHelper.exe	Ne	0.27 %
a1a1372..	trezord.exe	Ne	0.04 %
c67b7ac..	java.exe	Ne	0.12 %
709bc7f..	vulkaninfo.exe	Ne	5.30 %

Tabulka 16: Detekce legitimních aplikaci

9 Závěr

Tato diplomová práce předkládá základní poznatky o analýze malwaru a to jak statické, tak dynamické analýze. Zabývá se také základními parametry struktury spustitelného souboru. Dále shrnuje základní mechanismy obrany malwaru, které se snaží maskovat záměry a funkce škodlivého kódu. To je zásadní pro další vývoj analýzy malwaru, neboť je potřeba toto maskování detekovat potažmo i přes jeho přítomnost spolehlivě odhalovat malware. Dále se pak práce podrobněji zabývá analýzou statickou. Ke statické analýze pak nabízí přehled nejnovějších trendů v jejím využití, nezbytnou součástí je také výčet výhod a nedostatků tohoto konkrétního využití.

Na základě předloženého přehledu aktuálních znalostí na poli statické analýzy nabízí i vlastní řešení pomocí programu provádějícího statickou analýzu. Ten je dále testován, a to jak s vzorky legitimního softwaru, tak i se vzorky malwaru, přičemž důraz byl kladen na vzorky spustitelné v operačním systému s podporou formátu PE. Při analýze vlastním programem byly sledovány následující vlastnosti, jako jsou parametry spustitelného souboru, entropie, výskyt konkrétních skupin řetězců a nakonec výskyt konkrétních vzorů detekovaných pomocí nástroje Yara.

Výsledkem tohoto testování je soubor jednotlivých výstupů. Tyto výstupy jsou poté dále zkoumány.

Jako první byl vyhodnocován výstup parametrů spustitelného souboru. Legitimní software je běžně vyvíjen, jak pro 32 bitový, tak i 64 bitový operační systém a to zhruba v poměru 1:1, zatímco v případě malwaru převažuje výskyt 32 bitové varianty a to dokonce s 96 %. Dalším parametrem byl podpis souboru digitálním certifikátem. Zde se ukazuje, že 33 % běžných aplikací bylo podepsáno, naopak u malwaru to bylo 11 %. Byl zkoumán také typ PE souboru (dll knihovna nebo *exe* spustitelný soubor). Posledním pro analýzu zajímavým parametrem byl počet sekcí spustitelného souboru, kde u malwaru bylo nalezeno průměrně menší množství sekcí než u legitimního softwaru.

Jedním z dalších výstupů pak byla entropie. Získané hodnoty entropie pro legitimní software měly normální rozložení a medián entropie běžných aplikací byl 6.41. Hodnoty entropie malwaru pak nebyly rozloženy normálně a jejich medián byl 7.08. Pomocí Mann-Whitneyova U-testu pak byla otestována shodnost hodnot entropie pro legitimní software a malware. Rozdíl mezi těmito dvěma skupinami byl statisticky významný.

Následně byly vyhodnocovány řetězce. Konkrétně se jednalo o email, IP adresu (jak IPv4, tak IPv6) a URL adresu. Zajímavým zjištěním bylo, že v případě malwaru byl výskyt emailové a url adresy nižší. V případě malwaru mohou tyto informace, které slouží často ke komunikaci s útočníkem, zůstat skryté díky různým obfuskačním metodám.

Podobně jako u řetězců nebyl výstup hodnotící výskyt knihoven průkazný. Na datech totiž můžeme jasně vidět, že se použité knihovny u běžného softwaru i malwaru velmi podobají. Proto je potřeba zkombinovat tento výstup s výstupem obsahující seznam použitých metod, které jsou z těchto knihoven volány.

V případě výstupu funkcí byl pozorován znatelnější rozdíl mezi používanými metodami škodlivým kódem a běžnými aplikacemi. Malware častěji využíval metody, které slouží k dynamické práci s knihovnami a jsou často zneužívaný právě malwarem k injekci kódu.

Dále byl vyhodnocen výskyt použitých pravidel *Yara*. Nejčastěji bylo detekováno pravidlo pro detekci 32 bitového PE souboru, použití grafického rozhraní a práce se systémovými soubory. U malwaru pak bylo v 53 % detekováno packování a ve 44 % inicializace pro obsluhu strukturovaných výjimek. Dalším zajímavým pravidlem, které bylo detekováno u malwaru v 22 % byla snaha o získání vyššího oprávnění.

Tyto parametry byly následně také využity pro automatickou klasifikaci výstupu statické analýzy. Za tímto účelem byla využita technologie *ML.NET*.

Výsledky klasifikace malwaru nasvědčují, že úspěšnost použité detekce malwaru je velmi vysoká. U neškodlivého kódu pak byl v 15 % testovaných případů vyhodnocen testovaný legitimní software jako malware (false-positive). A proto by bylo vhodné datovou kolekci dále rozšířit o vzorky nové a to jak malwaru tak běžného softwaru. Zároveň je však neustále potřeba hledat nové možnosti a parametry (jako naříklad využití zmíněné obrazové analýzy malwaru), protože i obrana škodlivého kódu vůči analýze se nezastavuje.

Literatura

1. *Anti-VM and Anti-Sandbox Explained* [online]. Cyberbit, 2016 [cit. 2020-02-07]. Dostupné z: https://www.cyberbit.com/blog/endpoint-security/anti-vm-and-anti-sandbox-explained/.
2. SIKORSKI, Michael. *Practical Malware Analysis : a Hands-On Guide to Dissecting Malicious Software*. San Francisco: No Starch Press, 2012. ISBN 978-1-59327-290-6.
3. *Internet používá přes 80 % obyvatel Česka* [online]. Český statistický úřad, 2020 [cit. 2020-05-01]. Dostupné z: https://www.czso.cz/csu/czso/internet-pouziva-pres-80-obyvatel-ceska.
4. KOPÁČOVÁ, Nikola. *Počítače OKD napadli hackeři. Společnost přerušila těžbu ve všech dolech na Karvinsku* [online]. Český rozhlas, 2019 [cit. 2020-05-01]. Dostupné z: https://www.irozhlas.cz/zpravy-domov/okd-hackeri-doly-karvinsko_1912231724_pj.
5. SOLAŘÍKOVÁ, Ivana. *Výkupné se platí, vyjde totiž levněji, říká IT expert k útoku na nemocnici* [online]. MAFRA, a. s., 2020 [cit. 2020-05-01]. Dostupné z: https://www.idnes.cz/brno/zpravy/rozhovor-expert-martin-haller-hacker-kyberutok-fakultni-nemocnice-brno.A200314_061024_brno-zpravy_mos1.
6. *S kybernetickým útokem se v roce 2018 setkaly dvě pětiny velkých firem v ČR* [online]. Český statistický úřad, 2020 [cit. 2020-05-01]. Dostupné z: https://www.czso.cz/csu/czso/s-kybernetickym-utokem-se-v-roce-2018-setkaly-dve-petiny-velkych-firem-v-cr.
7. MELNICK, Jeff. *Top 10 Most Common Types of Cyber Attacks* [online]. Netwrix Corporation, 2018 [cit. 2020-05-01]. Dostupné z: https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/#Birthday%5C%20attack.
8. PRÁVO. *Sušická nemocnice se nachytala na podvodný e-mail, přišla o půl milionu* [online]. Borgis, a. s., 2020 [cit. 2020-05-01]. Dostupné z: https://www.novinky.cz/krimi/clanek/susicka-nemocnice-se-nachytala-na-podvodny-e-mail-prisla-o-pul-milionu-40315263.
9. *17 Phishing Prevention Tips – Prevent Phishing Attacks, Scams and Email Threats* [online]. DuoCircle LLC. [cit. 2020-05-01]. Dostupné z: https://www.phishprotection.com/content/phishing-prevention/.
10. CHRISTODORESCU, M.; JHA, S.; SESHIA, S. A.; SONG, D.; BRYANT, R. E. Semantics-aware malware detection. In: *2005 IEEE Symposium on Security and Privacy (S P'05)*. 2005, s. 32–46.
11. *Security report 2018/19* [online]. AV-TEST GmbH, 2019 [cit. 2020-05-01]. Dostupné z: https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2018-2019.pdf.

12. ZHANG, ELLEN. *What is Malware Analysis? Defining and Outlining the Process of Malware Analysis* [online]. Digital Guardian, 2017 [cit. 2020-03-16]. Dostupné z: https://digitalguardian.com/blog/what-malware-analysis-defining-and-outlining-process-malware-analysis.

13. A, Monnappa K. *Learning Malware Analysis: Explore the concepts, tools, and techniques to analyze and investigate Windows malware*. Packt Publishing, 2018. ISBN 1788392507. Dostupné také z: https://www.xarg.org/ref/a/1788392507/.

14. MALBOT. *Portable Executable File* [online]. 2019 [cit. 2020-03-22]. Dostupné z: https://malware.news/t/portable-executable-file/32980.

15. *List of file signatures: Wikipedia, The Free Encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 2020 [cit. 2020-03-23]. Dostupné z: https://en.wikipedia.org/wiki/List_of_file_signatures.

16. *How it works* [online]. VirusTotal, 2020 [cit. 2020-03-03]. Dostupné z: https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works.

17. LAB., Kaspersky. *Types of Malware* [online] [cit. 2020-03-24]. Dostupné z: https://usa.kaspersky.com/resource-center/threats/types-of-malware.

18. GENNARI, J.; FRENCH, D. Defining malware families based on analyst insights. In: *2011 IEEE International Conference on Technologies for Homeland Security (HST)*. 2011, s. 396–401. ISSN null. Dostupné z DOI: 10.1109/THS.2011.6107902.

19. *List of Executable File Extensions* [online]. 2019 [cit. 2020-03-20]. Dostupné z: https://www.lifewire.com/list-of-executable-file-extensions-2626061.

20. WIKIPEDIA. *Executable — Wikipedia, The Free Encyclopedia* [online]. 2020 [cit. 2020-03-20]. Dostupné z: https://en.wikipedia.org/wiki/Executable.

21. WIKIPEDIA. *Comparison of executable file formats — Wikipedia, The Free Encyclopedia* [online]. 2019 [cit. 2020-03-20]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Comparison%5C%20of%5C%20executable%5C%20file%5C%20formats&oldid=914596333.

22. PLACHY, Johannes. *Portable Executable File Format* [online]. 2018 [cit. 2020-03-20]. Dostupné z: https://blog.kowalczyk.info/articles/pefileformat.html.

23. ZATLOUKAL, Filip; ZNOJ, Jiří. Malware Detection Based on Multiple PE Headers Identification and Optimization for Specific Types of Files. *J. Adv. Eng. Comput.* 2017, roč. 1, s. 153–161.

24. *PE Format* [online]. 2019 [cit. 2020-03-18]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/debug/pe-format.

25. LIAO, Yibin. PE-Header-Based Malware Study and Detection. In: 2012.

26. REVERS3R. *Malware Researcher's Handbook (Demystifying PE File)* [online]. 2015 [cit. 2020-03-22]. Dostupné z: https://resources.infosecinstitute.com/2-malware-researchers-handbook-demystifying-pe-file/#gref.

27. CHOI, Y.; KIM, I.; OH, J.; RYOU, J. PE File Header Analysis-Based Packed PE File Detection Technique (PHAD). In: *International Symposium on Computer Science and its Applications*. 2008, s. 28–31. ISSN 2159-7049. Dostupné z DOI: 10.1109/CSA.2008.28.

28. YOU, Tuts 4. *PE iDentifier (PEiD) 0.95* [online]. 2008 [cit. 2020-03-22]. Dostupné z: https://tuts4you.com/e107_plugins/download/download.php?view.398.

29. A.S.L. *xeinfo PE by A.S.L - compression detector and data detector* [online]. 2020 [cit. 2020-03-22]. Dostupné z: http://www.exeinfo.xn.pl/.

30. CANNELL, Joshua. *Five PE Analysis Tools Worth Looking At* [online]. 2016 [cit. 2020-03-22]. Dostupné z: https://blog.malwarebytes.com/threat-analysis/2014/05/five-pe-analysis-tools-worth-looking-at/.

31. OCHSENMEIER, Marc. *pestudio: Malware Initial Assessment* [online]. 2020 [cit. 2020-03-22]. Dostupné z: https://www.winitor.com/index.html.

32. *Cracking – 2. část* [online]. 2006 [cit. 2020-03-18]. Dostupné z: http://programujte.com/clanek/2006080803-cracking-2-cast/.

33. *OllyDbg* [online]. 2014 [cit. 2020-03-18]. Dostupné z: http://www.ollydbg.de/.

34. *6 HEX Editors for Malware Analysis* [online]. 2010 [cit. 2020-03-18]. Dostupné z: https://www.sans.org/blog/6-hex-editors-for-malware-analysis/.

35. *Windows Sysinternals* [online]. 2017 [cit. 2020-03-18]. Dostupné z: https://www.chip.cz/casopis-chip/01-2017/windows-sysinternals/.

36. *Windows Sysinternals* [online]. Microsoft, 2019 [cit. 2020-03-18]. Dostupné z: https://docs.microsoft.com/en-us/sysinternals/.

37. *Dependency Walker 2.2* [online]. Dependency Walker, 2015 [cit. 2020-03-18]. Dostupné z: http://www.dependencywalker.com/.

38. *Common anti-debugging techniques in the malware landscape* [online]. 2017 [cit. 2020-03-18]. Dostupné z: https://www.deepinstinct.com/2017/12/27/common-anti-debugging-techniques-in-the-malware-landscape/.

39. *Thread Local Storage (TLS)* [online]. Microsoft, 2019 [cit. 2020-03-18]. Dostupné z: https://docs.microsoft.com/cs-cz/cpp/parallel/thread-local-storage-tls?view=vs-2019.

40. *CVE-2004-0733* [Available from IBM X-Force, CVE-ID CVE-2004-0733.]. 2004 [cit. 2020-03-16]. Dostupné z: https://exchange.xforce.ibmcloud.com/vulnerabilities/16711.

41. SINGH, Jagsir; SINGH, Jaswinder. Challenges of Malware Analysis: Obfuscation Techniques. *International Journal of Information Security Science*. 2018, roč. 7, č. 3, s. 100–110. ISSN 21470030. Dostupné také z: http://search.ebscohost.com/login.aspx?direct=true%5C&db=asn%5C&AN=133550405%5C&lang=cs%5C&site=eds-live.

42. LABS, DLT. *Introduction to Code Obfuscation* [online]. 2019 [cit. 2020-01-14]. Dostupné z: https://medium.com/better-programming/code-obfuscation-introduction-to-code-obfuscation-part-1-93a6797349b0.

43. *Code Obfuscation - Part 2: Obfuscating Data Structures* [online]. Paladion, 2005 [cit. 2020-01-14]. Dostupné z: https://www.paladion.net/blogs/code-obfuscation-part-2-obfuscating-data-structures.

44. BINIAM FISSEHA DEMISSIE Mariano Ceccato, Roberto Tiella. Assessment of data obfuscation with residue number coding. *Software Protection*. 2015, s. 38. Dostupné také z: http://search.ebscohost.com/login.aspx?direct=true%5C&db=edscma%5C&AN=edscma.2821440%5C&lang=cs%5C&site=eds-live.

45. *Code segment encryption* [online]. Sevagas, 2014 [cit. 2020-03-07]. Dostupné z: https://blog.sevagas.com/Code-segment-encryption.

46. *Code hardening (obfuscation & encryption)* [online]. Guardsquare, 2019 [cit. 2020-03-07]. Dostupné z: https://www.guardsquare.com/en/mobile-application-protection/code-hardening-obfuscation-encryption.

47. JOSEFSSON, S. *The Base16, Base32, and Base64 Data Encodings* [Internet Requests for Comments]. RFC Editor, 2006. ISSN 2070-1721. Dostupné také z: http://www.rfc-editor.org/rfc/rfc4648.txt. RFC. RFC Editor. http://www.rfc-editor.org/rfc/rfc4648.txt.

48. OCHODKOVÁ, Eliška. *Kryptografie a počítačová bezpečnost - Klasické algortimy - pokračování*. 2019.

49. GOLCHIKOV, Andrey Vladimirovich. *Executable file protection*. US 2002/0112158 A1.

50. ALBERTINI, Ange [online]. Stack Exchange, 2014 [cit. 2020-03-16]. Dostupné z: https://reverseengineering.stackexchange.com/a/1780.

51. HOON KANG, Brent Byung; SINCLAIR, Greg. Unpacking Malware. In: *Encyclopedia of Cryptography and Security*. Ed. TILBORG, Henk C. A. van; JAJODIA, Sushil. Boston, MA: Springer US, 2011, s. 1350–1351. ISBN 978-1-4419-5906-5. Dostupné z DOI: 10.1007/978-1-4419-5906-5_851.

52. *Packers*. Ange Albertini, 2010. Dostupné také z: https://gironsec.com/code/packers.pdf.

53. ARNTZ, Pieter. *Explained: Packer, Crypter, and Protector* [online]. 2017 [cit. 2020-03-22]. Dostupné z: https://blog.malwarebytes.com/cybercrime/malware/2017/03/explained-packer-crypter-and-protector/.

54. YAN, W.; ZHANG, Z.; ANSARI, N. Revealing Packed Malware. *IEEE Security Privacy*. 2008, roč. 6, č. 5, s. 65–69. ISSN 1558-4046. Dostupné z DOI: 10.1109/MSP.2008.126.

55. MILKOVIC, Marek. *Generic Unpacker of Executable Files* [online]. 2015 [cit. 2020-03-24]. Dostupné z: http://excel.fit.vutbr.cz/submissions/2015/030/30.pdf.

56. *x86 Instruction Set Reference* [online]. 2020 [cit. 2020-03-08]. Dostupné z: https://c9x.me/x86/html/file_module_x86_id_45.html.

57. *Registration Authority: Standard Group MAC Address* [online]. IEEE [cit. 2020-04-25]. Dostupné z: https://standards.ieee.org/products-services/regauth/grpmac/index.html.

58. *nsmfoo/antivmdetection* [online]. GitHub, 2019 [cit. 2019-12-10]. Dostupné z: https://github.com/nsmfoo/antivmdetection.

59. IJAZ, Muhammad; DURAD, Muhammad Hanif; ISMAIL, Maliha. Static and Dynamic Malware Analysis Using Machine Learning. *2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Applied Sciences and Technology (IBCAST), 2019 16th International Bhurban Conference on*. 2019, s. 687–691. ISSN 978-1-5386-7729-2. Dostupné také z: http://search.ebscohost.com/login.aspx?direct=true%5C&db=edseee%5C&AN=edseee.8667136%5C&lang=cs%5C&site=eds-live.

60. SUN, Bowen; LI, Qi; GUO, Yanhui; WEN, Qiaokun; LIN, Xiaoxi; LIU, Wenhan. Malware family classification method based on static feature extraction. *2017 3rd IEEE International Conference on Computer and Communications (ICCC), Computer and Communications (ICCC), 2017 3rd IEEE International Conference on*. 2018, s. 507–513. ISSN 978-1-5090-6352-9. Dostupné také z: http://search.ebscohost.com/login.aspx?direct=true%5C&db=edseee%5C&AN=edseee.8322598%5C&lang=cs%5C&site=eds-live.

61. BAKOUR, Khaled; UNVER, H. Murat; GHANEM, Razan. The Android Malware Static Analysis: Techniques, Limitations, and Open Challenges. *2018 3rd International Conference on Computer Science and Engineering (UBMK), Computer Science and Engineering (UBMK), 2018 3rd International Conference on*. 2018, s. 586–593. ISSN 978-1-5386-7893-0. Dostupné také z: http://search.ebscohost.com/login.aspx?direct=true%5C&db=edseee%5C&AN=edseee.8566573%5C&lang=cs%5C&site=eds-live.

62. MOSER, A.; KRUEGEL, C.; KIRDA, E. Limits of Static Analysis for Malware Detection. *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*. 2008, s. 421–430. ISSN 978-0-7695-3060-4. Dostupné také z: http://search.ebscohost.com/

login.aspx?direct=true%5C&db=edseee%5C&AN=edseee.4413008%5C&lang=cs%5C&site=eds-live.

63. NARAYANAN, Barath Narayanan; DJANEYE-BOUNDJOU, Ouboti; KEBEDE, Temesguen M. Performance analysis of machine learning and pattern recognition algorithms for Malware classification. *2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS), Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS), 2016 IEEE National*. 2017, s. 338–342. ISSN 978-1-5090-3441-3. Dostupné také z: http://search.ebscohost.com/login.aspx?direct=true%5C&db=edseee%5C&AN=edseee.7856826%5C&lang=cs%5C&site=eds-live.

64. BALTHROP, J.; FORREST, S.; NEWMAN, M.; WILLIAMSON, M. Technological networks and the spread of computer viruses. 2004. Dostupné také z: http://search.ebscohost.com/login.aspx?direct=true%5C&db=edsarx%5C&AN=edsarx.cs%5C%2f0407048%5C&lang=cs%5C&site=eds-live.

65. GANDOTRA, Ekta; BANSAL, Divya; SOFAT, Sanjeev. Malware Analysis and Classification: A Survey. *Journal of Information Security*. 2014, roč. 05, č. 02, s. 56–64. Dostupné z DOI: 10.4236/jis.2014.52006.

66. *PeNET* [online] [cit. 2020-04-19]. Dostupné z: https://github.com/secana/PeNet.

67. *VirusTotal.NET - A full implementation of the VirusTotal 2.0 API* [online] [cit. 2020-04-19]. Dostupné z: https://github.com/Genbox/VirusTotalNet/.

68. *How it works* [online] [cit. 2020-04-19]. Dostupné z: https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works.

69. *ML.NET* [online]. Microsoft [cit. 2020-04-25]. Dostupné z: https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet.

70. *Yara - The pattern matching swiss knife for malware researchers (and everyone else)* [online] [cit. 2020-04-19]. Dostupné z: http://virustotal.github.io/yara/.

71. *yarGen* [online]. Github [cit. 2020-04-25]. Dostupné z: https://github.com/Neo23x0/yarGen.

72. *libyara.NET* [online] [cit. 2020-04-19]. Dostupné z: https://github.com/microsoft/libyara.NET.

73. *Mime* [online]. Github [cit. 2020-04-25]. Dostupné z: https://github.com/hey-red/Mime.

74. HOLEC, Miroslav. *Jak na .NET Standard knihovny a generování NuGet balíčků* [online]. 2018 [cit. 2020-04-19]. Dostupné z: https://www.miroslavholec.cz/blog/jak-na-net-standard-knihovny-a-generovani-nuget-balcku.

75. BUCHANAN, Bill. *Entropy* [online]. Networksims [cit. 2020-04-25]. Dostupné z: https://asecuritysite.com/encryption/ent.

76. *API responses: /file/scan* [online]. VirusTotal [cit. 2020-04-25]. Dostupné z: https://developers.virustotal.com/reference#file-scan-upload-url.

77. *Hardware Survey - OS Marketshare* [online]. PassMark Software [cit. 2020-04-25]. Dostupné z: https://www.pcbenchmarks.net/os-marketshare.html.

78. *GetModuleHandle* [online]. aldeid [cit. 2020-04-25]. Dostupné z: https://www.aldeid.com/wiki/GetModuleHandle.

79. ARVANAGHI, Brandon. *DLL Injection Using LoadLibrary in C* [online] [cit. 2020-04-25]. Dostupné z: https://arvanaghi.com/blog/dll-injection-using-loadlibrary-in-C/.

80. BANACH, Zbigniew. *What Is Privilege Escalation and Why Is It Important?* [online]. 2019 [cit. 2020-04-25]. Dostupné z: https://www.netsparker.com/blog/web-security/privilege-escalation/.

81. *Cross-validation (statistics) — Wikipedia, The Free Encyclopedia* [online]. Wikipedia, 2020 [cit. 2020-04-25]. Dostupné z: https://en.wikipedia.org/wiki/Cross-validation_(statistics).

82. *How to choose an ML.NET algorithm* [online]. Microsoft, 2019 [cit. 2020-04-25]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/machine-learning/how-to-choose-an-ml-net-algorithm.

A Příloha v IS EDISON

Součásti vypracování této diplomové práce je příloha, jež obsahuje vypracované řešení popsané v kapitole 8. Tento projekt byl implementován pomocí Visual Studio 2019.

Řešení je rozděleno do několika projektů:

- *StaticAnalysisProject.Lib* - knihovna;
- *StaticAnalysisProject.Console* - konzolová aplikace;
- *StaticAnalysisProject.Web* - webová aplikace;
- *StaticAnalysisProject.BuildTraningSet* - program pro vytvoření datového setu;
- *StaticAnalysisProject.UpdateTrainingSet* - program pro aktualizaci datového setu;
- *StaticAnalysisProject.MalwareStats* - program, který byl vytvořen při vytváření statistik;
- *MalwareSample.BotnetListener* - klient pro síť botnet;
- *MalwareSample.Keylogger* - keylogger;
- *MalwareSample.RemoteData* - jednotlivé skripty pro vzdálené ovládání vytvořeného malwaru;
- *DP.Tests* - testovací program.

B Jednotlivé hashe

Jednotlivé kontrolní součty, které byly zkráceny v kapitole 8.4.

B.1 Malware

cbfc9d3bbb07b9594226898510bc36121ac6c7b3ba97a7007bba88ab724ee86f
Nedetekováno
875708802b4c0c29c188335d6cf3c11567184cb03a6649bbcabf762fd0eab423
Nedetekováno
000ad60fb69e0b29422ff2f87b6baaa94d9554f7749c52e22535ee7f61fafe80
Win32:PUP-gen [PUP]
037bfec3b29e3eaa9fae89efce8bca7aaf17c5c2ce1237400d03b6b911224d85
Win32:AdwareSig [Adw]
0872a6c5f7f9454adcbc632dd4557ccc953ebe34e58233cb9a50b9729ceaa0d6
Win32:VBCrypt-AKV [PUP]
190ea43bb2ea29a7d68ee4c40ad4997a4c676e12d9a1399108a597449520dc86
Win32:Downloader-GYZ [Trj]
1c757b4194a5d4e69752eae25d1e1bff2420ba781e0ee3a528315b1e254b4999
Win32:Adware-CWL [PUP]
21b0957a16999e82f4de677d42c9a8726f4f18fa0a8d8d0513dc766a0a5d5149
Win32:Adware-gen [Adw]
2891ae420367fbdecbd7d3ddfc153f0e00ee4acd9676b6b63920ed7ec4f98358
Win32:Adware-gen [Adw]
2e58e15136c7f34fd8a9664e7cf1baf82fb3b873d60dba2065d0e05e16449693
Win32:AdwareSig [Adw]
436b7f21b04faaaeb165aa260eca676db5e9941f0dde88c10fd71d6d121cfdc8
Win32:Vundo-GZ [Trj]
56bb2f51bb18c0e830e757cc615b9d53bbd54657f8b184a70786db1d547ee047
Win32:AdwareSig [Adw]
78c38d998fcbbae770f16294b2b9508a748a253c8a10087f72ccb1a632bb078b
Win32:Dropper-OYB [PUP]
97f1adcebecb103618abcd3959137b588a233433344131fd96dccb7b862eb671
Win32:AdwareSig [Adw]
a2cb313a78898f45d21465fa2a4b5ae17dfeb7510bffaf10e7b6866074a6ed0d
Win32:RmnDrp
abaa67c6622ebf51bc95437bf1260881639c6a92bbdf2933c4a757e4476ecd0b
Win32:AdwareSig [Adw]
b154f80ea1db013c4642a1fb15819470e76ea6ffd1e033c105d2df9ed63ed8ea
Win32:AdwareSig [Adw]

bf5b8849ca74ede846597efd54b8be4e17d88c2b22bf4ab8115805b657ab0ae1
Win32:Mirc-AA [PUP]
c494dab0118d3afbf4b8fda81d3b23cd612c03cc4afe693696fe62f96d6e2c19
Win32:Malware-gen
db1ca5abda08c0c7012d4acac0cf9d3ba65af9ab93d93ec88311f76eac1a583b
Win32:Malware-gen

B.2 Software

85601e92142ef3034b259d823926555252cd620bfc0b474736d3267a4c886f8e
Ozone Neon M50 Driver.exe
9299fb2782866eec592a2b096ec7fbdaf323b588d645fd4831b4547380b80d0b
Curse.exe
baf1879a3dd327ca06937febbdd611f2041acb61285242b939fa782f4508a364
Lanayo.VagrantManager.exe
775a767251f270c25205e81d0c7a09bdbe74861ae3db14168261ff2e26c73df9
Amazon.Fuel.AppCore.dll
dcdb0650b7bcb619e8309cd9871af05ac83722311f80e84dc96701b509ee2d32
CAMV2.Data.dll
459dd1efe7457b94263b4c45658b87126982e3f9106d34fd86890e2f1d158c72
EasyAntiCheat.exe
47772a978a07fdc1681a6fe0174f00c80b95661d8cfcb6679105d27cb8e955e1
NeroPatentActivation.exe
f2d6f5b9d167659aebd7b7ca43906de0d238a19632e9d8ff1350c9688e2050ee
lua52.dll
c1c6c4c8de6b0fac66382fe4d619f23149fa7f722aba71f305c9d553a176bc9c
mergecap.exe
280c0f7b28874b9c9bb68780f62376bc137d64f9b1583c4d1354ee7cfb1df0a1
obs32.exe
10effd4181e639696ba94731b9abf8823f0658f8b50a1a9490ab05fd4f229cdc
OpenRL.dll
3ad5d5bcfe61eefc94adbc3ea0beec3d4131be4c77d749df61e5ed955a01d9b4
overlay.dll
b3e7e2ee3b5232d76f5e0b949f363adff83333c6dd73b6206aaa8f8595c1b00f
phpstorm.exe
483863541fda0db28df3befe559e484d64376de0e702c0dc90f886eeb8f3905b
python.exe
f4aae235166395e0d3bc1de45e76b9a6ce0387d98c7453c446f6b42b3ff9057b
Qt5Core.dll
a64fe801def20f64fa97c6bea6dd9893c35dfa4d384be6867abec9bd41ff7c1d

socialclub.dll
41229db9d328818a51623b61e41b79be3058641d4e2fdf8f97063e481589f788
SocialClubHelper.exe
a1a1372544627ead59633cfd6aa25e9c007b1a9edfaa3e99c3543f961440c323
trezord.exe
c67b7acce03210d67a9eac3125acd59815917e6be8e042a353934c0f76d784ae
java.exe
709bc7fd42cadae44712140027e59bae062e9be24fa57849d7321922bca65ff6
vulkaninfo.exe